
175 PTAS
61
mi computer
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
9 788485 822836 10061
Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VI-Fascículo 61

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-034-7 (tomo 6)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 138503
Impreso en España - Printed in Spain - Febrero 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

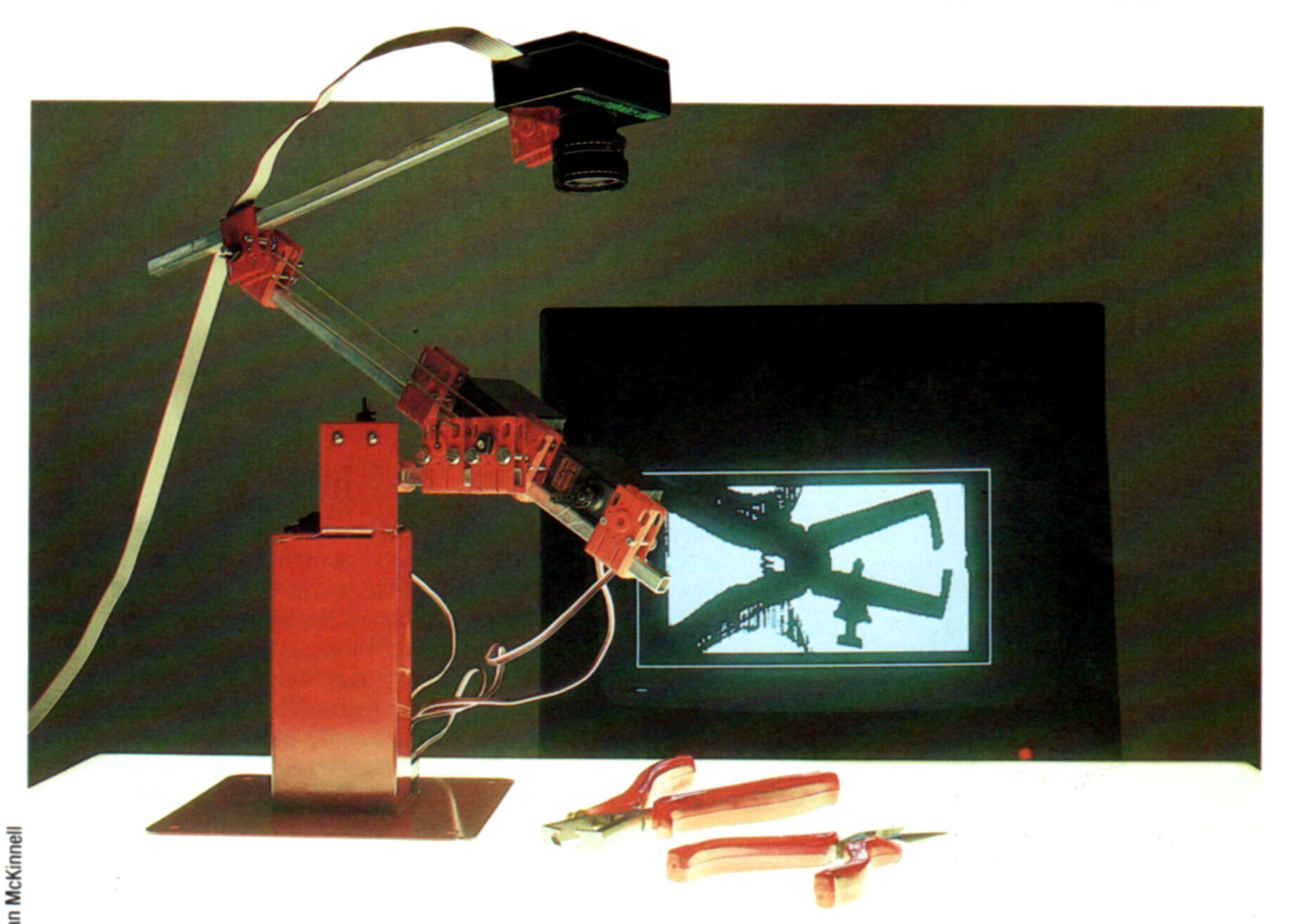
No se efectúan envíos contra reembolso.





Superar obstáculos

Ha llegado el momento de que estudiemos cómo conseguir que un robot se mueva de una manera auténticamente "inteligente"



Ian McKinnell

Construyendo una imagen
Si sus procesadores son suficientemente sofisticados, los robots móviles pueden aprender un catálogo de objetos arquetípicos para utilizar con algoritmos para reconocimiento de formas y comparación de patrones. El brazo-robot Beasty, que vemos en la fotografía, está equipado con una cámara Snap, que produce una imagen digital, y el software Snap, que incluye un módulo para reconocimiento de objetos. Una vez que el objeto ha sido "visto" desde distintos ángulos, el brazo tiene una razonable posibilidad de reconocerlo en cualquier posición

En primer lugar, no queremos controlar al robot mediante un operador humano. Si el operador debe observarlo y controlar hasta el más mínimo de sus movimientos, entonces en muchas aplicaciones la utilización de un robot no tendría el menor sentido: la persona bien podría llevar a cabo las tareas que realiza éste. Esto, por supuesto, no es así para todas las situaciones. Los robots que se utilizan para desactivar bombas son controlados por el hombre, porque para guiarlos correctamente aún sigue siendo necesaria la experiencia humana.

Asimismo, tampoco tiene mucho sentido controlar un robot a través de una secuencia fija de instrucciones almacenadas en un ordenador. Esto resultaría ser poco más que un autómata, un dispositivo que sigue al pie de la letra la secuencia incorporada, independientemente de las circunstancias. Nuevamente hay ocasiones en la que tales dispositivos son útiles: con frecuencia los brazos-robot se consideran "inteligentes" aun cuando lleven a cabo un conjunto preprogramado de acciones.

Sin embargo, hemos definido al robot "inteligente" como aquel que pueda servir una taza de café por la mañana. Este robot no podría ser controlado por un ser humano, dado que su función consiste en llevar a cabo su tarea cuando aún no hay ningún ser humano despierto. Si este dispositivo servidor de café se programara con una secuencia fija de instrucciones, surgirían problemas si se cambiara de lugar la cama o se dejara ropa en el suelo.

De modo que nuestra definición de movimiento inteligente es la capacidad de un robot de desplazarse por su entorno sin que lo controle ningún ser humano y sin seguir ciegamente ninguna secuencia fija de instrucciones. Debería ser capaz de moverse de un punto a otro, evitando cualquier obstáculo.

En el campo de la inteligencia artificial existe la tradición de utilizar juegos de una u otra clase para examinar problemas complejos de esta clase. Así como los programas para jugar al ajedrez han permitido profundizar considerablemente en otras ramas de la inteligencia artificial, los robots que corren a través de laberintos pueden ayudarnos en nuestra definición de movimiento auténticamente inteligente. A fines de los años setenta comenzaron en Estados Unidos las competiciones de "microrratones", y en 1980 se celebró en Gran Bretaña la primera competición de esta naturaleza. La idea era muy sencilla: se construyó un gran laberinto de alrededor de tres metros cuadrados y los participantes hubieron de diseñar "ratones-robot" que pudieran descubrir por sí mismos el camino hasta el centro sin ninguna clase de ayuda. El laberinto se componía de pequeños cuadrados de igual tamaño cuyos lados en ocasiones estaban abiertos para mostrar una posible ruta y otras veces cerrados

Kevin Jones



Tan sencillo como el ABC
En la resolución de laberintos funcionan algoritmos simples. Este robot avanza por espacios vacíos hasta que llega a un callejón sin salida: los cuadrados F, I y P, por ejemplo. Entonces retrocede hasta la última intersección que encontró (en este caso el cuadrado G) marcando todos los cuadrados implicados como inútiles en su mapa mental. Si desde una intersección no hay ninguna ruta todavía no probada, el robot retrocede hasta la intersección anterior a ésta y así sucesivamente, todo el camino de regreso hasta la entrada, si fuera necesario, en cuyo caso el laberinto es "ciego" o insoluble

Solución al laberinto

```
  49 REM.....CBM 64..........
  50 REM.SOLUCION AL LABERINTO.
  51 REM.....CBM 64..........
 100 GOSUB 2000             :REM INICIO
 150 GOSUB 9000             :REM IMPR.LABERINTO
 200 FOR L=1 TO 1
 250 GOSUB 7000             :REM MIRAR ALREDEDOR
 300 GOSUB 8000             :REM MOVERSE
 400 NEXT L
 450 FI=16:CO=12:GOSUB 9500
 500 IF LAB < > HM THEN PRINT"CIEGO"
 550 IF LAB=HM THEN PRINT"CASA"
 900 PRINT:PRINT:INPUT A$:RUN
1999 REM ***********************
2000 REM *       INICIO        *
2001 REM ***********************
2100 TM=10:S2=TM*TM:GX=TM/2:GY=TM+2
2120 DIM LAB(TM,TM),R$(4),LX(4),LY(4)
2140 X=RND(-TI)
2150 DEFFNR(N)=INT(RND(1)*N+1)
2160 HM=-1:G0=-2:WL=42:W$=CHR$(WL)
2180 CL$=CHR$(147):H$=CHR$(19)
2200 X=GX-1:Y=GY-2:DR=3
2220 D$=CHR$(17):P$=D$
2240 FOR K=1 TO 5:P$=P$+P$:NEXT K:P$=H$+P$
2400 DATA 0,1,"0",-1,0,".",0,-1," ",1,0," "
2420 FOR K=1 TO 4:READ LX(K),LY(K),R$(K)
2490 NEXT K:RETURN
6999 REM ***********************
7000 REM *  MIRAR ALREDEDOR    *
7001 REM ***********************
7110 FI=1:CO=1:GOSUB9500
7120 L=0:ND=0:FOR S=1 TO 4
7140 NX=X+LX(S):NY=Y+LY(S)
7200 IF NX < 1 OR NX > TM THEN NEXT S:RETURN
7220 IF NY < 1 OR NY > TM THEN NEXT S:RETURN
7230 LAB=LAB(NX,NY):IF LAB=0 THEN ND=S
7260 IF LAB=HM THEN ND=S:S=4:L=1
7280 NEXT S:RETURN
7999 REM......................
8000 REM. MOVERSE            .
8001 REM......................
8100 LAB(X,Y)=DR:FL=1
8120 FI=Y+1:CO=X+1:GOSUB 9500
8140 IF ND=0 THEN ND=DR-2-4*(DR<3):FL=2
8160 PRINT R$(FL):X=X+LX(ND):Y=Y+LY(ND)
8180 DR=ND: IF Y>TM THEN L=1:RETURN
8200 IF FL=2 THEN DR=LAB (X,Y)
8490 RETURN
8999 REM ***********************
9000 REM *  IMPRIMIR LABERINTO *
9001 REM ***********************
9040 PRINT CL$;:FI=1:CO=1
9050 WL=42:W$=CHR$(WL)
9060 S$="                     "
9070 FOR K=1 TO TM+2:T$=T$+W$:NEXT K
9080 E$=W$+LEFT$(S$,TM)+W$
9100 PRINT T$:FOR J=2 TO TM+1
9120 PRINT E$:NEXT J:PRINT T$
9140 FOR K=1 TO S2/2
9150 WX=FNR(TM):WY=FNR(TM)
9160 LAB(WX,WY)=WL:FI=WY+1:CO=WX+1
9200 GOSUB 9500:PRINT W$:NEXT K
9250 CO=1+FNR(TM):FI=1+FNR(TM):GOSUB 9500
9300 PRINT"H":LAB(CO-1,FI-1)=HM
9350 FI=GY:CO=GX:GOSUB9500:PRINT" "
9490 RETURN
9499 REM......................
9500 REM.  POSICIONAR CURSOR  .
9501 REM......................
9600 PRINT LEFT$(P$,FI)TAB(CO-1);:RETURN
```

Complem. al BASIC

Introduzca las siguientes modificaciones en este programa:

BBC Micro

```
  49 REM *************** BBC ***************
  50 REM *      SOLUCION AL LABERINTO       *
  51 REM *************** BBC ***************
9040 CLS:FI=1:CO=1
9600 PRINT TAB(CO-1,FI-1);:RETURN
```

Spectrum

```
  49 REM *********** SPECTRUM ***********
  50 REM *     SOLUCION AL LABERINTO     *
  51 REM *********** SPECTRUM ***********
2120 DIM LAB(TM,TM):DIM R$(4)
2130 DIM LX(4):DIM LY(4)
2140 RANDOMIZE
2150 DEFFNR(N)=INT(RND*N+1)
9040 CLS:FI=1:CO=1
9600 PRINT AT (FI-1,CO-1);:RETURN
```

Brian Morris

para denotar una pared. El ratón que llegó al centro en el menor tiempo ganó la competición.

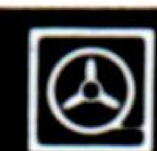
En el primer certamen de microrratones celebrado en Gran Bretaña sólo hubo cinco participantes. Algunos se comportaban de una forma sumamente errática; uno ni siquiera podía avanzar en línea recta y hasta el mejor de los ratones quedaba bastante confundido después de dar la vuelta a un par de esquinas. En el mismo año se celebraron las finales europeas de esta competición y participaron ratones procedentes de Finlandia, Suiza y Alemania. Finalmente hubo uno que consiguió recorrer el laberinto correctamente; se trataba del *Stirling mouse* de Nick Smith, que estaba equipado con sencillos sensores mecánicos que corrían a lo largo de la parte superior de las paredes del laberinto y funcionaba con un sencillo motor paso a paso. Desde entonces, el interés en esta clase de competiciones ha aumentado, y en la Euromouse Contest que se celebró en Madrid en 1984 el mejor tiempo conseguido para alcanzar el centro del laberinto fue de 31,4 segundos. Algunos participantes tampoco entonces pudieron llegar a la meta, pero la mayoría sí lo logró.

El mapa del laberinto

¿Y cómo recorre un ratón-robot un laberinto? En general, el robot debe usar un método preciso para su desplazamiento, de modo que sepa exactamente cuál es su posición en cualquier momento dado; esto se puede lograr montando al robot sobre ruedas y activándolo con motores paso a paso, a menudo utilizando alguna forma de realimentación interna, como codificadores de eje. El robot requiere, asimismo, un conjunto de sensores para detectar la presencia o la ausencia de paredes, lo cual le permite construir un "mapa" del laberinto. En las competiciones de microrratones a los robots se les concede un par de vueltas de entrenamiento, que utilizan para elaborar un plan del recorrido. Luego efectúan la vuelta de competición, durante la cual se cronometran sus intentos por alcanzar el centro.

Si bien los métodos exactos varían de un robot a otro, una respuesta es equipar por delante al robot con un sencillo detector táctil. Colocado en el centro de cada cuadrado del laberinto, puede verificar si tiene alguna pared exactamente delante suyo. Luego gira 90º en el sentido de las agujas del reloj, vuelve a verificar y repite la secuencia. Finalmente "sabrá" dónde están todas las paredes de cada cuadrado del laberinto. Esta información se puede almacenar como un único número binario de cuatro bits; de modo que 1111 en binario representaría un cuadrado con paredes en sus cuatro lados (imposible en la práctica, porque en ese cuadrado el robot ni siquiera podría entrar), y 0000 correspondería a un cuadrado sin ninguna pared. 0111 representaría entonces un cuadrado con tres paredes y una abertura: un callejón sin salida.

Esta información se podría retener en una matriz bidimensional; en BASIC se podría utilizar DIM A (16,16) para representar un laberinto con 16 "celdas" en cada dirección. El robot entonces ha de calcular una ruta que lo conduzca hasta A(8,8), si se considera que ése es el centro del laberinto. A menudo el robot posee un programa de ordenador incorporado que calcula una estructura de árbol para cada ruta a través del laberinto. Muchas de las ramas del árbol conducirán a callejones sin salida o



llevarán al robot de regreso hasta un punto que ya ha visitado; en estos casos las bifurcaciones se "podan" y no se tienen en cuenta. El programa busca entonces en las bifurcaciones restantes para hallar la ruta con menos cuadrados. Entonces adopta ese camino como su ruta hacia el centro.

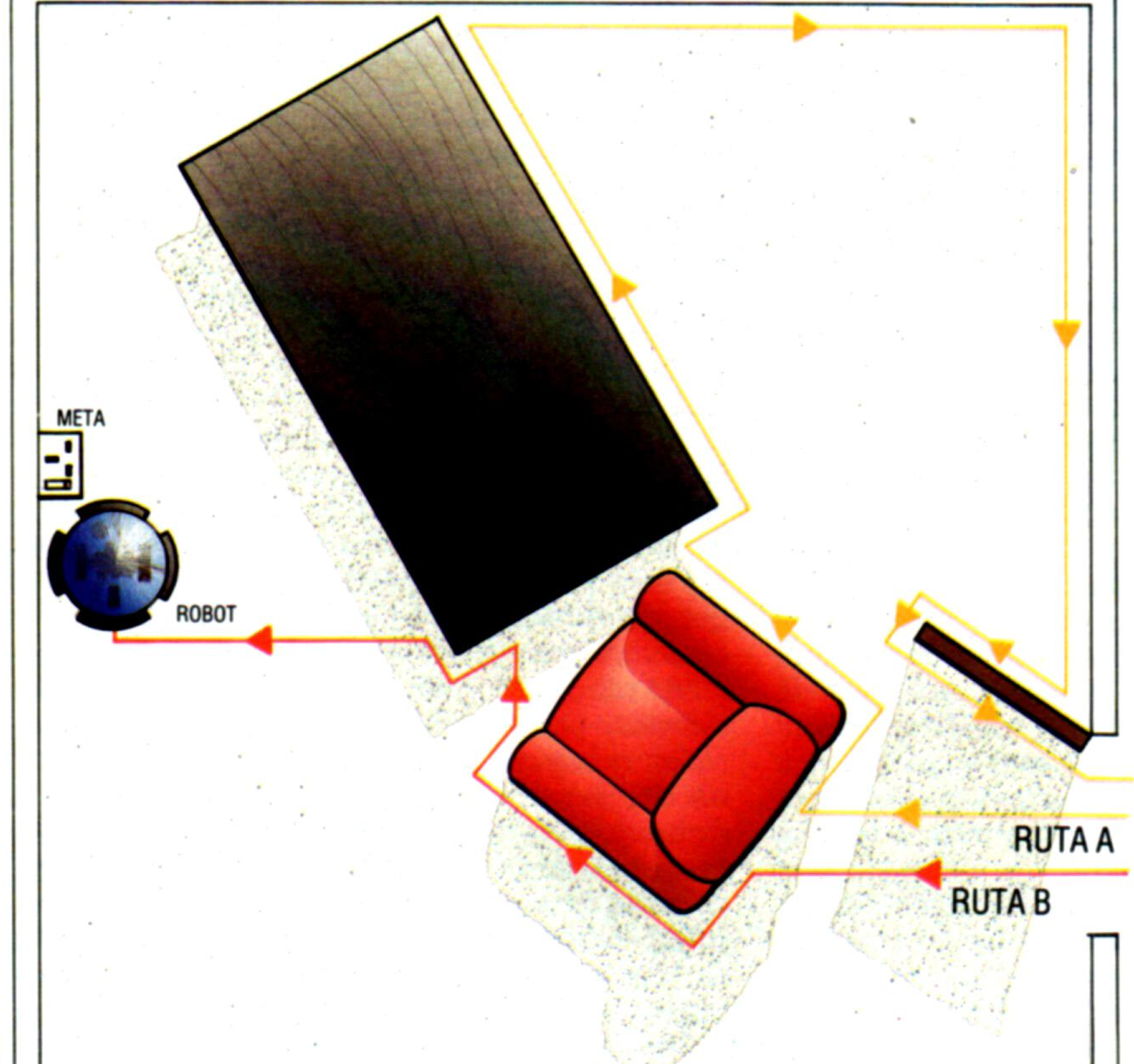
Se puede adaptar este método para que proporcione una estrategia más eficaz. Los sensores del robot son esenciales para su éxito. Por ejemplo, sensores táctiles mecánicos sencillos exigen que el robot choque realmente contra cada una de las paredes para trazar su mapa; los sensores de proximidades pueden detectar una pared sin tocarla y un sensor de distancia puede detectar la posición de una pared al final de un camino largo y despejado a través del laberinto. Obviamente, equipar al robot con cuatro sensores en vez de uno le permitiría "mirar" en las cuatro direcciones a la vez y eliminaría la necesidad de hacerlo girar en cada cuadrado.

Por la casa

De modo que vemos que un robot puede actuar "inteligentemente" cuando recorre un laberinto. En muchos sentidos, el problema de construir un robot que pueda hallar su camino a través de su casa es muy similar. El robot debe utilizar sensores para calcular las posiciones de todos los objetos de una habitación, y debe luego planificar una ruta que lo lleve, sorteando todos los obstáculos, hasta su punto de destino. Los problemas adicionales que implica este tipo de movimiento inteligente provienen del hecho de que el diseño de una habitación es muchísimo más complejo que el diseño de un laberinto. Una habitación corriente no está dividida simplemente en cuadrados, ni todos sus componentes permanecen fijos en el mismo lugar. El robot servidor de café podría aprender la posición de varios objetos; pero si se moviera una silla o si el gato se sentara en el suelo, el robot debería entonces modificar el camino elegido.

Este problema sólo se puede resolver haciendo que el robot haga un uso continuo de sus sensores para actualizar su mapa interno. El problema del gato exige mayor atención porque, dado que los robots no saben nada de gatos (como tampoco, si vamos al caso, de personas), al robot le resulta difícil calcular lo que debe hacer en su primer encuentro con un felino. (Qué duda cabe de que el gato tendrá el mismo problema cuando se encuentre por primera vez con un robot.) La mejor solución es equipar al robot con un sensor de movimiento, que es un sensor de distancia que responde a la medición de distancias variables y, por consiguiente, puede hacer frente a objetos en movimiento. Una vez detectado algo que se está desplazando, lo mejor que puede hacer el robot es permanecer inmóvil hasta que el objeto deje de moverse o se aleje. Puede ser que esto no parezca muy inteligente, y por cierto es menos cordial que acercarse al gato y acariciarlo, pero una acción de este tipo es muy similar a la reacción de muchos animales, que se quedan estáticos cuando detectan objetos en movimiento.

El tema del movimiento inteligente va, por consiguiente, ligado al empleo de sensores conjuntamente con un programa de ordenador. Un robot sin sensores no será capaz de moverse inteligentemente y, mientras más sensores tenga instalados, mejor será su conocimiento del mundo. Es este conocimiento lo que le permite al robot mostrar signos de inteligencia.



Kevin Jones

Por dos caminos
Abrirse paso a través de objetos desconocidos nunca es fácil. La ruta A muestra la huella de un sencillo robot doméstico tratando de hallar el enchufe de la corriente. Su único algoritmo para evitar objetos (conocido como *seguimiento de pared*) es seguir los bordes de las cosas mientras sus sensores de corto alcance buscan el objeto. Este método primitivo puede ser muy eficiente en entornos sencillos, pero trampas y escollos como los que vemos en la ilustración pueden hacerlo fracasar.
La ruta B muestra la huella de un robot similar con un algoritmo ligeramente mejorado: cuando ha de girar un cierto ángulo al hallar un objeto, prefiere hacerlo describiendo el menor ángulo posible, dado que ello reducirá la cantidad de "rastreo hacia atrás" que realice. Esta sencilla modificación reduce en gran medida su vulnerabilidad a las trampas y permite que sus sensores de exploración controlen su comportamiento con mayor eficacia

Programa modelo

Esta vez analizaremos el "Abacus", el paquete de hoja electrónica que se suministra con el Sinclair QL



MENÚ DE MENSAJES



MENÚ DE INSTRUCCIONES

Gran parte del interés que generó el Sinclair QL se ha centrado en los cuatro paquetes de software que se suministran junto con la máquina: *Quill* (tratamiento de textos), *Archive* (base de datos), *Easel* (programa para gráficos) y *Abacus* (hoja electrónica). Estos paquetes contienen algunos elementos de diseño integrado (véanse pp. 982-983). Se puede transferir datos entre ellos; los modelos de hoja electrónica, por ejemplo, se pueden visualizar como gráficos o incorporar en un documento preparado mediante *Quill*. Las pantallas de visualización son similares y algunas instrucciones son comunes a todos los paquetes: por ejemplo, tres de las cinco teclas de función del QL dan resultados idénticos en las cuatro aplicaciones (F1 es la tecla Help, F2 controla la zona de "mensajes" de la parte superior de la pantalla y F3 llama a las instrucciones). No obstante, los programas se deben cargar y ejecutar por separado.

Hay dos formas diferentes de cargar el *Abacus* (de hecho, cualquiera de los paquetes del QL). La primera supone colocar el cartucho en el microdrive 1 y después pulsar F1 para seleccionar la opción pantalla o F2 si se utiliza un televisor para la visualización. Los paquetes del QL incluyen rutinas para la carga automática de los programas *(boot)*. Por el contrario, si ya se ha seleccionado la pantalla, entre `lrun mdv1_boot` (suponiendo que *Abacus* esté en la unidad 1) y aparecerá la pantalla inicial.

La pantalla muestra la porción superior izquierda de la matriz de la hoja electrónica; en su documentación, Psion, la firma fabricante, alude a la misma como una "cuadrícula". Inicialmente, se visualizan las columnas de la A a la F y las filas del 1 al 15, si bien las dimensiones máximas de la cuadrícula del *Abacus* son de 64 columnas y 255 filas. (Compare estas dimensiones con las dimensiones máximas de la cuadrícula del *Vu-Calc*, de 28 columnas y 55 filas.) En la parte superior de la visualización de la cuadrícula hay un conjunto de mensajes, y debajo de ella hay una línea para entrada de datos, junto con información que refleja el estado del modelo en curso. Los mensajes se pueden eliminar (pulsando F2), pero son muy útiles para los principiantes, dado que explican en cada momento las opciones disponibles. Ésta es, en realidad, una zona de "menú" que indica qué teclas de función se utilizan para controlar diversas operaciones y que muestra cómo mover el cursor o ir directamente hasta una celda determinada, cómo entrar datos o texto y cómo llamar a las instrucciones. Sin embargo, no es un auténtico menú: las opciones no se pueden seleccionar posicionando el cursor sobre la opción correspondiente, sino que el usuario las debe digitar.

La utilización del *Abacus* para tareas básicas es muy simple, si bien para un tipo de tareas más avanzado llevará un tiempo acostumbrarse a algunas instrucciones y expresiones. El siguiente ejemplo, basado nuevamente en un presupuesto doméstico, ilustrará la forma de trabajar del *Abacus*.

En primer lugar, necesitamos un encabezamiento general. Al igual que con el *Vu-Calc*, las entradas de textos deben ir precedidas por comillas. A nuestro modelo lo llamaremos PREVISIÓN DE EFECTIVO, y pulsando la tecla de cursor apropiada, pasamos a la celda D1 y simplemente entramos unas comillas seguidas del texto. El *Abacus*, al igual que la mayoría de las hojas electrónicas, permite que el texto exceda de una celda si la adyacente está vacía, de modo que es fácil entrar hasta los títulos más largos.

También podemos subrayar el encabezamiento, mejorando de ese modo el aspecto de nuestro modelo. Para hacerlo debemos desplazar el cursor hasta la celda de debajo de nuestro título (D2) y entrar `rept ("=",len(d1))`. Aquí `rept` es el equivalente de la instrucción `REPLICATE` del *Vu-Calc*, y = le dice al programa qué símbolo utilizar (estamos utilizando el signo "igual" como un subrayado doble). El resto de la instrucción (`len(d1)`) es la forma de decirle a *Abacus* que repita el símbolo para toda la longitud del texto de la celda D1.

A diferencia del *Vu-Calc*, que posee una anchura de columna fija de nueve caracteres, *Abacus* nos permite seleccionar anchuras distintas para diferentes aplicaciones. En nuestro caso necesitamos que la columna A sea más ancha, para que haya espacio suficiente para entrar texto de longitud variable, y además precisamos que las otras columnas sean más estrechas, de modo que se puedan visualizar los datos para seis meses. Para hacer esto, utilizamos la tecla de función F3, seguida de `G` (para seleccionar la instrucción `GRID`, cuadrícula) y `W` (para la instrucción `WIDTH`, anchura). La línea de entrada indicará que la anchura actual es 10. Nosotros deseamos cambiar la anchura de la columna A a 15, de modo que tecleamos `15` en respuesta al mensaje. El programa ahora nos solicitará una serie de celdas a las cuales se aplicará la nueva anchura. Entrando `A` y `A` como los dos parámetros indicamos que sólo la columna A debe poseer esta anchura. Luego debemos volver a pasar por el mismo procedimiento, esta vez seleccionando una anchura de 6 y una serie de B a G. Con esto obtenemos lugar suficiente para visualizar las cifras de seis meses.

Ahora hemos de entrar etiquetas para las columnas de "meses". Ello se puede hacer simplemente digitando el texto correspondiente en cada celda, pero *Abacus* posee una facilidad especial para los nombres de los meses. Desplace el cursor hasta A3 y digite `row=month(col()−1)` (fila=mes...). La línea de entrada solicitará ahora una serie entre `B` y `G` en respuesta a los mensajes, y las columnas se etiquetarán automáticamente. Hay nombres de meses demasiado largos como para caber en nuestras columnas ajustadas, por lo cual hay que abreviarlos. Esto

se realiza volviendo a digitarlos, recordando colocar las comillas para indicar texto.

El siguiente paso consiste en entrar encabezamientos para las diversas filas, desplazando el cursor hacia abajo una línea después de cada entrada (lamentablemente, *Abacus* no lo proporciona como una facilidad automática). Nuestro modelo da por supuesto que el cabeza de familia es un vendedor cuyos ingresos se componen tanto de un salario básico como de comisiones. El modelo permite calcular los ingresos en base a comisiones sobre las ventas previstas más el salario básico. Las ventas reales conseguidas se pueden entrar a medida que vayan transcurriendo los meses, y se pueden entonces realizar previsiones revisadas para los meses futuros. Los valores negativos se visualizan entre paréntesis en la cuadrícula terminada. Los encabezamientos a entrar son éstos: Ventas: previstas y reales; Comisión; Salario básico; Ingresos totales; Gastos: hipoteca, impuestos, agua, electricidad, gas, teléfono; Gastos totales; Líquido positivo (o negativo); Balance bancario de apertura y cierre.

Después de haber entrado los encabezamientos de columnas y filas, podemos empezar a rellenar con cifras nuestra hoja electrónica. Primero entraremos las previsiones de ventas para los seis meses visualizados en la pantalla. Éstas pueden ser, por ejemplo, 200 000, 240 000, 260 000, 220 000, 240 000 y 300 000 ptas, y se deben entrar sin el punto. Las ventas reales no se pueden entrar todavía, de modo que proseguiremos calculando la comisión sobre las ventas previstas. La misma se calcula como el 20 % de las ventas que superen las 200 000 ptas, de modo que la fórmula a entrar en la celda B10 es row=(ventas−200000)*.2. Nuevamente la serie requerida es de B a G; entre usted estas letras en respuesta a los mensajes y la comisión se calculará y se visualizará al instante.

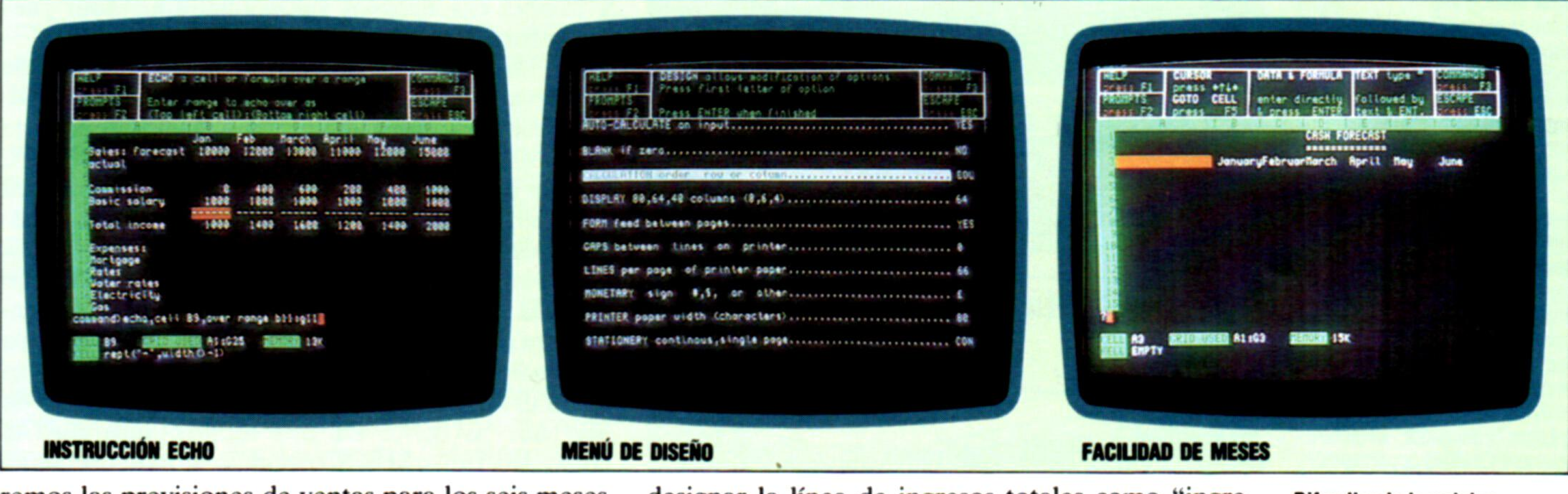
Si el salario básico es de 34 000 ptas mensuales, éste se puede entrar en la celda B11 como row=34000, otra vez para la serie de B a G. Ahora se puede digitar la fórmula para ingresos totales en B13. La misma es row=sum(col) con una serie de filas de 0 a 11 y una serie de columnas de B a G. Esto completa el cálculo de los ingresos, pero se puede mejorar la presentación del modelo agregando rayas arriba y abajo de la fila de "ingresos totales". Para hacer esto, desplace el cursor hasta B12 y entre row=rept("−",width()−2). *Abacus* responderá produciendo cuatro guiones debajo de cada cifra de "salario básico". Para llevar a cabo la misma operación en la fila 14, produciendo por consiguiente guiones arriba y abajo de la fila de "ingresos totales", se utiliza la instrucción ECHO(eco). Con el cursor aún en B9, el sistema pedirá una serie sobre la cual reproducir el subrayado: responda con B14:G14. (Si se olvidó de dejar filas libres cuando entró los encabezamientos, éstas se pueden insertar empleando la instrucción GRID.) Para completar el formateado, se debe "justificar por la derecha" todo el subrayado utilizando la instrucción J para la serie de B2 a G14. Ello asegurará que todos los guiones queden alineados con las cifras.

Ahora ya se pueden entrar los gastos, utilizando la fórmula de fila (*row*) para cifras mensuales estándares tales como la hipoteca, o simplemente entrando cada cifra en las celdas correspondientes para pagos irregulares, como pueden ser la electricidad y el gas. Los gastos totales se pueden, entonces, definir como la suma de todas estas cifras, de la misma forma que se calcularon los ingresos totales.

Del mismo modo, la cantidad líquida positiva o negativa se puede calcular como los ingresos totales menos los gastos totales. Lamentablemente, *Abacus* parece ignorar cualquier parte de una instrucción después de un espacio, de modo que no podemos entrar simplemente row=ingresos totales−gastos totales. Debemos, por consiguiente, volver a designar la línea de ingresos totales como "ingresos" y entrar row=ingresos−total, nuevamente para la serie de B a G. Ahora aparecerán en la pantalla las entradas y las salidas líquidas para cada mes.

El paso final consiste en calcular los balances bancarios. Entre primero el balance inicial: un descubierto de 50 000 ptas, por ejemplo. Entre esta cantidad como −50000. Ahora calcule el balance de cierrre, utilizando la fórmula row=neto+apertura con el cursor en B28, para la ya familiar serie de B a G. Ello producirá el balance de cierre de enero. Los balances de apertura para los otros meses se calculan entrando en C27 la fórmula row=B28. Para hacer este trabajo para todas las columnas, primero es necesario cambiar el orden de cálculo de fila a columna utilizando la instrucción DESIGN. Entonces se calcularán todos los balances de apertura y cierre y se volverán a calcular de inmediato si se modificara cualquier cifra. Tal como está, nuestro modelo refleja los balances negativos como cifras precedidas por un signo menos. Para cambiar este formato de modo que una cifra negativa se indique mediante paréntesis, utilizamos la instrucción UNITS y respondemos al mensaje con B.

Ahora nuestro modelo ya está completo. Se puede salvar seleccionando la instrucción SAVE con un nombre de archivo adecuado. Después se puede imprimir o se pueden entrar cifras distintas.



Difundiendo la palabra
Un atractivo diseño de pantalla, buenas características y una amplia facilidad de mensajes/ayudas hacen que el *Abacus* sea una hoja electrónica agradable y fácil de utilizar. Aquí mostramos la instrucción ECHO en acción copiando una fila a otra, las opciones de la instrucción DESIGN para formatear la hoja electrónica y la facilidad *month* (meses), que genera los nombres de los meses con la sola pulsación de una tecla. También se puede apreciar la pantalla por defecto con el menú de mensajes (*prompts*) arriba, y la hoja en blanco mostrando el menú de instrucciones (*commands*)

En busca de nombres

Ya estamos en condiciones de ampliar la utilidad de búsqueda de variables para que incluya la facilidad de cambio de nombres

Nuestro programa para reemplazar variables es una utilidad más compleja que la simple búsqueda de nombres de variables que desarrollamos en las páginas 1144 y 1180. Por este motivo necesitamos añadir un programa en código máquina. La CPU 6502 del BBC Micro y la CPU 6510 del Commodore 64 poseen el mismo lenguaje assembly, de modo que es una buena idea analizarlas juntas.

Nuestra primera tarea consiste en encontrar un método de retener dos programas separados en la memoria del ordenador al mismo tiempo. Como ya hemos explicado, en el BBC Micro podemos hacerlo alterando las variables incorporadas en BASIC PAGE e HIMEM. En el Commodore 64 necesitamos un programa en código máquina para alterar los diversos apuntadores de la memoria de página cero. La primera parte del listado en lenguaje assembly, que comienza en la etiqueta SWITCH, hará esto por nosotros.

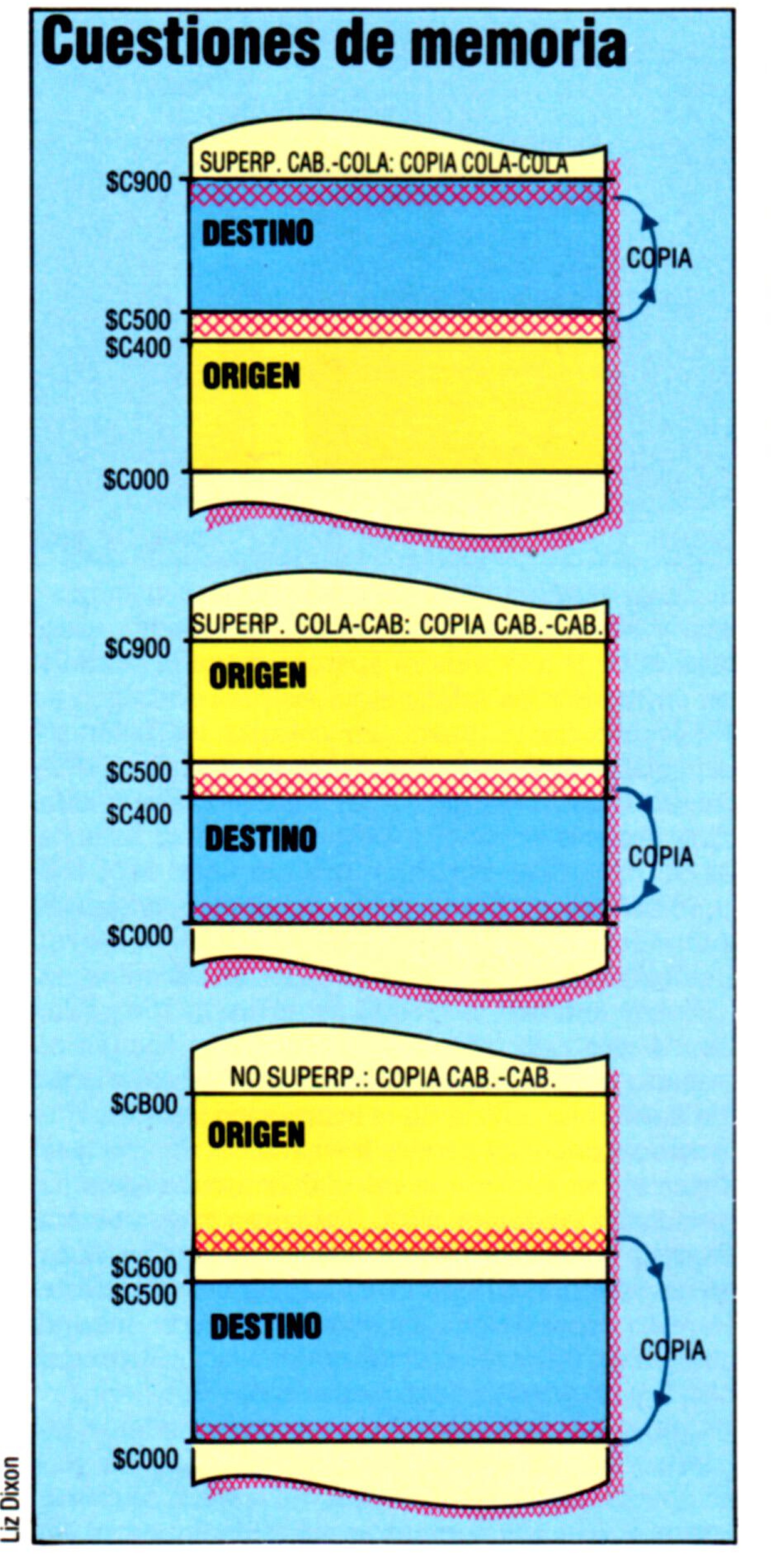
La rutina SWITCH nos permitirá acomodar los programas en BASIC: uno empieza en la dirección 800 hexa (el sitio habitual para un programa en BASIC) y el otro comienza en la dirección 9000 hexa. SWITCH comienza por mirar el apuntador TXTTAB para ver cuál de las zonas de programa es la actual, y cambia entonces los valores del apuntador para convertir en actual la otra zona de programa.

TXTTAB se cambia para que apunte al principio de la nueva zona de programa, luego FRETOP y MEMSIZ deben apuntar al byte que sigue al último byte de la nueva zona de programa, mientras FRESPC apunta al final de la nueva zona de programa. El programa busca entonces en la cadena de apuntadores de direcciones de enlace (véase p. 1184) para hallar el final del programa en BASIC, utilizando VARTAB como apuntador temporal. Cuando encuentra los dos bytes de dirección de enlace a cero que indican el final del programa, incrementa dos veces el apuntador anterior y copia el resultado en ARYTAB y STREND. De este modo, tanto VARTAB como ARYTAB y STREND apuntan al byte que sigue inmediatamente después al programa en BASIC.

Los principales cambios que hemos de introducir en el programa en BASIC son las subrutinas extras de las líneas 30500 y 30600. La primera de éstas encuentra el final del programa en BASIC, utilizando los bytes de longitud de línea en la versión para el BBC y los apuntadores de la siguiente línea en la versión para el Commodore 64.

El cambio en los nombres de variables lo efectúa realmente la subrutina de la línea 30600. Cuando el nombre antiguo de la variable y el nombre nuevo tienen la misma longitud, el nuevo nombre simplemente se puede escribir encima del antiguo. Cuando la longitud del nombre antiguo y la del nombre nuevo son distintas, el procedimiento es algo más complicado. En este caso, el programa debe o bien hacer espacio extra, o reducir todo el espacio innecesario en el programa que esté modificando, y efectuar las modificaciones correspondientes en las variables que utilice para llevar el registro de su posición en el programa que está alterando. Asimismo, debe cambiar el byte de longitud de línea en la versión para el BBC y los apuntadores a la línea siguiente en la versión para el Commodore 64.

El programa también emplea subrutinas en código máquina para crear o suprimir espacio. Aunque esto se podría hacer en BASIC, sería inaceptablemente lento incluso en un programa de tamaño



Copia inteligente
La rutina de sustitución ha de desplazar grandes secciones del programa en BASIC hacia arriba y hacia abajo de la memoria cuando inserta nombres de variables de diferente longitud. Al hacerlo satisface las cuatro posibles condiciones de las direcciones de origen y de destino. Si siempre copiara desde el principio del bloque origen, entonces, cuando la cabeza del bloque de destino se superpusiera con la cola del bloque origen, lo que copiara se escribiría encima de algunos de los datos del origen. Una rutina de copia "inteligente" detectará este caso y evitará la corrupción copiando este bloque origen a partir de la cola. Una rutina de copia "tonta" siempre copia a partir de la cabeza del bloque origen



medio, con millares de bytes a desplazar cada vez.

Hay dos subrutinas en lenguaje máquina: UP para hacer espacio extra y DOWN para suprimir el espacio innecesario. Los detalles del bloque de programa a desplazar se pasan al código máquina a través de las posiciones de memoria CURR, LAST y DIFF.

Para la subrutina UP es necesario inicializar las siguientes posiciones de memoria:

CURR: dirección de la parte inferior del bloque a desplazar
LAST: uno menos que la dirección de la parte superior del bloque a desplazar
DIFF: cantidad de bytes a liberar

y para DOWN es necesario inicializarlas así:

CURR: dirección de la parte superior del bloque a desplazar
LAST: uno menos que la dirección de la parte inferior del bloque a desplazar
DIFF: cantidad de bytes a obtener

Observe que las dos subrutinas empiezan en extremos opuestos del bloque a desplazar y realizan sus desplazamientos en sentidos contrarios. Esto es para evitar que los datos se sobreescriban antes de haberlos desplazado.

Para utilizar la versión BBC del programa de sustitución de variables, primero debe entrarse (o cargarse, LOAD) y luego ejecutar (RUN) el programa en assembly para poner en la memoria el código máquina. Después cargue (LOAD) el programa a modificar y digite:

P%=PAGE

Esto pasa la dirección de comienzo del programa al programa de sustitución de variables. Luego ponga en PAGE un valor por encima de LOMEM, y cargue (LOAD) y ejecute (RUN) el programa de sustitución de variables. Puede volver al programa modificado mediante:

HIMEM=PAGE−1
PAGE=P%

Para utilizar la versión del programa para el Commodore 64, también necesitará poner en la memoria el código máquina, ya sea con el programa cargador de BASIC o bien ensamblando el código fuente. Si ensambla el código fuente, o carga el código máquina directamente como código máquina, tendrá que colocar (POKE) ceros en las direcciones 36864, 36865 y 36866. Esto equivale a renovar (NEW) la zona para programas alternativa.

Después de cargar el código máquina, SYS 49152 cambiará de una zona de programa a la otra. Si olvida dónde está, puede averiguarlo mediante PRINT PEEK(44), que dará 8 para la zona de programas normal y 144 para la zona de programas alternativa.

Una vez que tenga en el ordenador el código máquina, puede cargar (LOAD) el programa a modificar en la zona de programas normal y el programa para sustitución de variables en la zona de programas alternativa, ejecutando (RUN) luego el programa de sustitución de variables.

Conmutación y copia C64

```
  10 DIR=49152
  20 FOR LINEA=1000 TO 1180 STEP 10
  30 S=0
  40 FOR DIR=DIR TO DIR+7
  50 READ BYTE
  60 POKE DIR, BYTE
  70 S=S+BYTE
  80 NEXT DIR
  90 READ SUMACONTROL
 100 IF S<> SUMACONTROL THEN PRINT"ERROR DE DATOS EN
     LINEA";LINEA:END
 110 NEXT LINEA
 120 POKE 36864,0:POKE36865,0:POKE36866,0
1000 DATA162,0,164,44,192,8,208,9,787
1010 DATA160,160,32,72,192,169,144,208,1137
1020 DATA7,160,144,32,72,192,169,8,784
1030 DATA162,1,134,43,133,44,134,45,696
1040 DATA133,46,160,0,177,45,170,200,931
1050 DATA177,45,224,0,208,240,201,0,1095
1060 DATA208,236,230,45,208,2,230,46,1205
1070 DATA136,16,247,165,45,133,47,133,922
1080 DATA49,165,46,133,48,133,50,96,720
1090 DATA134,51,132,52,134,55,132,56,746
1100 DATA202,136,134,53,132,54,96,160,967
1110 DATA0,177,251,164,255,145,251,160,1403
1120 DATA0,196,251,208,2,198,252,198,1305
1130 DATA251,165,253,197,251,208,234,165,1724
1140 DATA254,197,252,208,228,96,164,255,1654
1150 DATA177,251,160,0,145,251,230,251,1465
1160 DATA208,2,230,252,165,253,197,251,1558
1170 DATA208,236,165,254,197,252,208,230,1750
1180 DATA96,0,0,0,0,0,0,96
```

Copia BBC

```
 10 MODE 7
 20 HIMEM=HIMEM−&100
 30 CURR = &70
 40 LAST = &74
 50 DIFF = &78
 60 FOR PASS = 1 TO 2
 70 P%=HIMEM
 80[ OPT 1
 90 .UP          LDY #0
100 .UP1         LDA (CURR),Y
110              LDY DIFF
120              STA (CURR),Y
130              LDY #0
140              CPY CURR
150              BNE UP2
160              DEC CURR+1
170 .UP2         DEC CURR
180              LDA LAST
190              CMP CURR
200              BNE UP1
210              LDA LAST+1
220              CMP CURR+1
230              BNE UP1
240              RTS
250 .DOWN        LDY DIFF
260              LDA (CURR),Y
270              LDY #0
280              STA (CURR),Y
290              INC CURR
300              BNE DOWN1
310              INC CURR+1
320 .DOWN1       LDA LAST
330              CMP CURR
340              BNE DOWN
350              LDA LAST+1
360              CMP CURR+1
370              BNE DOWN
380              RTS
390]
400 NEXT PASS
410PRINT~UP,~DOWN
```

Sustitución de variables

Commodore 64

Introduzca estas modificaciones en el programa de p. 1180:

```
30005 INPUT"REEMPLAZAR POR";RS
30006 CURR=251
30007 LAST=253
30008 DI=255
30035 GOSUB 30500
30036 IF ERR THEN PRINT"NO ENCUENTRO EL
      FINAL DEL PROGRAMA":END
30060 IF SIGLINEA=0 THEN END
30065 CURRLINE TEXTPTR
Eliminar la línea 30085
30460 IF NOMBRE$=T$ THEN GOSUB 30600
30465 IF ERR THEN PRINT"NO HAY LUGAR
      EN LA LINEA";NUMLINEA:END
```

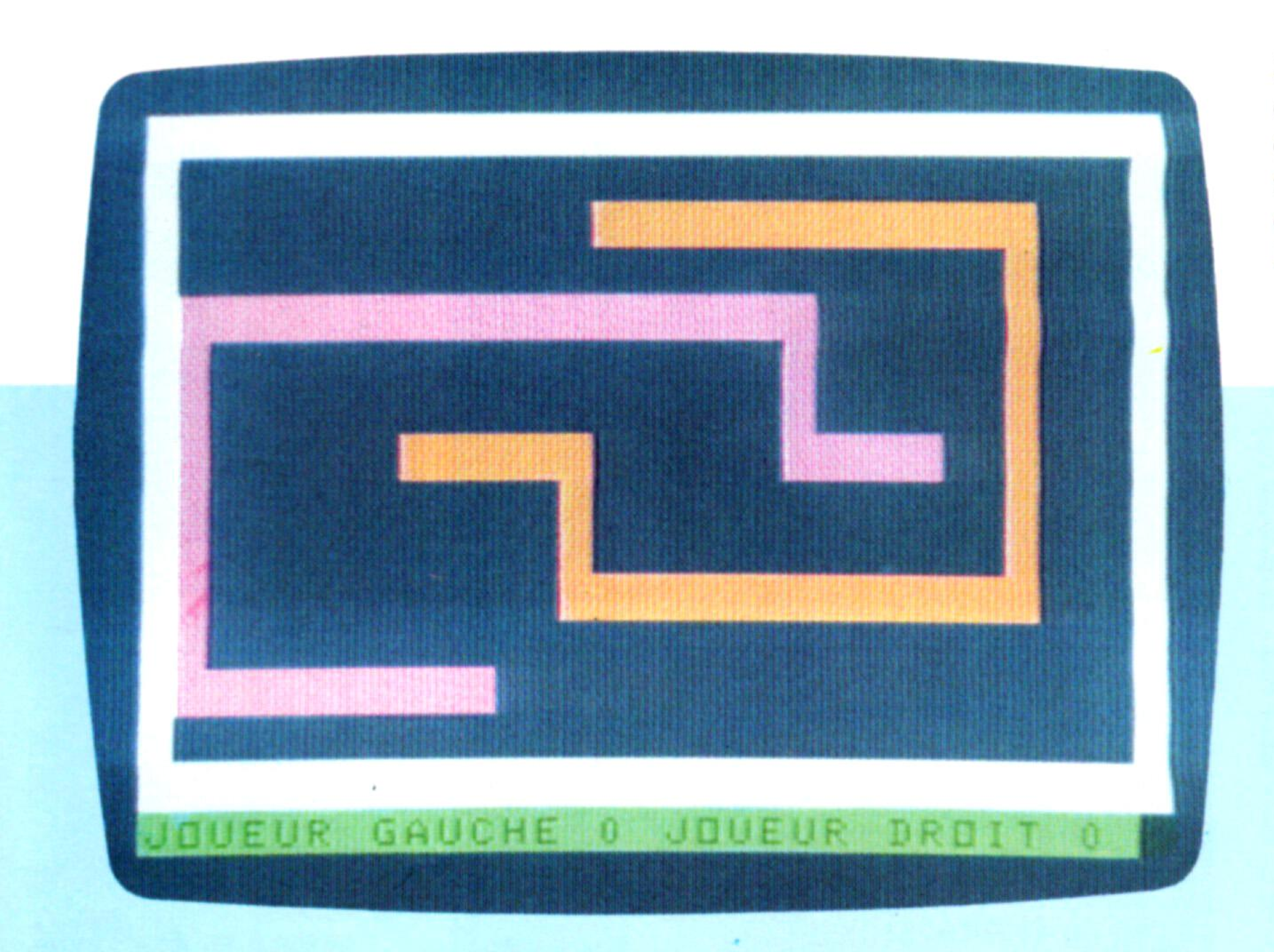
BBC Micro

Introduzca estas modificaciones en el programa de p. 1145:

```
30005 INPUT"Reemplazar por";
      SUSTITUCION$
30006 CURR = &70
30007 LAST = &74
30008 DIFF = &78
30055 GOSUB 30500
30056 IF Outofroom THEN PRINT"No encuentro
      el final del programa":PRINT"PAGE en
      lugar equivocado?":END
30060 Apuntadortexto=P%
30070 IF Numlinea > 32767 THEN END
30105 Bytelongitud=Apuntadortexto
30460 IF Nombre$=Objetivo$ THEN GOSUB
      30600
30465 IF Outofroom THEN PRINT"No hay lugar
      en la linea";Numlinea:END
```

Trazos

¿Una guerra de trincheras? No: este juego nos propone más bien una auténtica guerra de líneas. He aquí una versión para el microordenador Dragon



Dos jugadores se enfrentan para dividirse el espacio vital. Cada uno de ellos debe esforzarse, al irse desplazando, por no cortar jamás su trazado o el de su adversario y por no salirse del rectángulo dibujado en la pantalla. Utilice las palancas de mando o las siguientes teclas:

Jugador de la derecha: P, L, ; y .

Jugador de la izquierda: W, A, S y Z

```
10 REM*********
20 REM *TRAZOS*
30 REM *********
40 GOSUB 850
50 GOSUB 630
60 ON JK GOTO 150
70 FOR I=1 TO 50
80 NEXT I
90 D$=INKEY$
100 C1=(D$="L")-(D$=";")+32*((D$="P")-
    (D$="."))
110 C2=(D$="A")-(D$="S")+32*((D$="W")
    -(D$="Z"))
120 IF C1<>0 THEN D1=C1
130 IF C2<>0 THEN D2=C2
140 GOTO 270
150 K0=JOYSTK (0)
160 K1=JOYSTK (1)
170 K2=JOYSTK (2)
180 K3=JOYSTK (3)
190 IF K0<26 AND K1>37 THEN S1=-1
200 IF K0>37 AND K1<26 THEN S1=1
210 IF K0<26 AND K1<26 THEN S1=-32
220 IF K0>37 AND K1>37 THEN S1=32
230 IF K2<26 AND K3>37 THEN S2=-1
240 IF K2>37 AND K3<26 THEN S2=1
250 IF K2<26 AND K3<26 THEN S2=-32
260 IF K2>37 AND K3>37 THEN S2=32
270 IF S1<>0 THEN D1=S1
280 IF S2<>0 THEN D2=S2
290 P1=P1+D1
300 IF PEEK (P1)<>128 THEN 360
310 POKE P1,J1
320 P2=P2+D2
330 IF PEEK (P2)<>128 THEN 410
340 POKE P2,J2
350 GOTO 60
360 F2=F2+1
370 GOSUB 590
380 IF F2=10 THEN 460
390 D$=INKEY$
400 GOTO 50
410 F1=F1+1
420 GOSUB 590
430 IF F1=10 THEN 500
440 D$=INKEY$
450 GOTO 50
460 CLS
470 PRINT@ 165, "GANA JUGADOR IZQUIERDA"
480 PRINT@ 202,F2;"A";F1
490 GOTO 530
500 CLS
510 PRINT@ 165, "GANA JUGADOR DERECHA"
520 PRINT@ 202,F1;"A";F2
530 R$=INKEY$
540 PRINT@266, "OTRA?"
550 R$=INKEY$
560 IF R$=" " THEN 540
570 IF R$<>"N" THEN RUN
580 END
590 FOR I=5 TO 255 STEP 5
600 SOUND I,1
610 NEXT I
620 RETURN
630 CLS 0
640 C=207
650 J1=239
660 J2=255
670 P1=1272
680 P2=1256
690 D1=-1
700 D2=1
710 FOR I=1024 TO 1055
720 POKE I,C
730 POKE I+448,C
740 NEXT I
750 FOR I=1 TO 13
760 POKE I*32+1024,C
770 POKE I*32+1055,C
780 NEXT I
790 PRINT@ 480, "JUGADOR IZQUIERDA";F2,
    "JUGADOR DERECHA";F1;
800 POKE P1,J1
810 POKE P2,J2
820 S1=0
830 S2=0
840 RETURN
850 CLS
860 PRINT@ 170, "PALANCAS MANDO?"
870 D$=INKEY$
880 IF D$=" " THEN 870
890 JK=-(D$="S")
900 RETURN
```

Visión periférica



He aquí un repaso a algunos conocidos periféricos y útiles indicaciones que el usuario debería tener en cuenta antes de adquirirlos

Hasta hace poco tiempo, el periférico más importante para el usuario de un ordenador personal era un módulo enchufable que contenía memoria extra. La memoria era cara y máquinas como el ZX-81 y el Vic-20 se diseñaron para mantener los costes al mínimo; de hecho, el ZX-81 no ampliado le ofrecía al usuario tan sólo setecientos y pico bytes con los que escribir programas. Las máquinas de hoy en día ya vienen equipadas con hasta 64 K de RAM (algunas, como el QL, ofrecen todavía más) y, por consiguiente, en la actualidad raramente se requieren "RAM *packs*". El usuario de hoy día tiene multitud de periféricos entre los que escoger: los modems permiten la comunicación entre usuarios que viven a cientos de kilómetros de distancia el uno del otro, los vehículos motorizados o los brazos-robot se pueden controlar utilizando una interface adecuada, y se pueden emplear sintetizadores de voz para entretenimiento o con fines educativos.

Si bien existen en el mercado muchos dispositivos periféricos diferentes, la mayoría sólo se fabrican para las máquinas más vendidas. Ésta es una consideración que se debe tener presente al decidir qué máquina adquirir; los usuarios de un Spectrum, un Commodore y un Acorn siempre tendrán más elecciones que quienes opten por un Oric o un Sord. La creación de un mercado de periféricos para las máquinas más modernas lleva su tiempo, si bien el recientemente introducido estándar MSX hará que las cosas resulten más fáciles al permitir que todas las máquinas que respondan a las especificaciones MSX utilicen los mismos accesorios.

Una consideración primordial para el comprador de periféricos es el tema de la compatibilidad: todo lo que se compre debe funcionar con cualquier otro dispositivo que se pueda llegar a adquirir en el futuro. El ejemplo clásico en este sentido es el del Spectrum. Muchos usuarios de esta máquina han comprado la Interface Uno y uno o dos microdrives, sólo para descubrir que algunos de los periféricos que ya poseían (e incluso parte de su software) no funcionan teniendo instalada la Interface Uno.

Sin embargo, de mantenerse la compatibilidad entre dispositivos, la elección de accesorios para su máquina puede incrementar en gran medida el atractivo que posee la informática. Periféricos adecuados pueden permitirle diseñar un sistema de ordenador que se adapte a sus propias exigencias particulares, y este sistema se puede ir modificando luego a medida que sus necesidades evolucionen.

Sistemas de almacenamiento

El dispositivo de almacenamiento más común para utilizar con un microordenador es la habitual grabadora de cintas de cassette de audio corrientes. Ésta ofrece la ventaja de ser sencilla de utilizar y relativamente económica, pero sus inconvenientes se hacen evidentes enseguida. Cargar programas ocupa mucho tiempo y es difícil llevar el registro exacto de qué es lo que hay en una cinta determinada. Las unidades de disco son más rápidas y más fiables, pero cuestan más. La mayoría de los ordenadores personales están limitados a un tipo de unidad de disco, y algunas de éstas (en particular los modelos de Commodore) son notablemente lentas en operación. La mayor parte de las máquinas personales continúan utilizando unidades de 5 1/4", pero ahora se están popularizando más las unidades de 3 o de 3 1/2". La unidad de disco Oric/Atmos, por ejemplo, emplea discos de 3" con una capacidad de 160 K por cada lado. El BBC Micro es sumamente flexible, ya que permite que se le conecten muchos sistemas de disco diferentes. El Torch Disk Pack convierte verdaderamente al BBC en un ordenador nuevo, con un microprocesador Z80 para complementar al 6502 del ordenador y 64 K de memoria adicionales. El Torch proporciona, asimismo, cuatro programas de "gestión" y viene con una versión del BASIC BBC que está diseñada para trabajar con el procesador Z80.
El Sinclair Spectrum, por el contrario, no posee medios para conectar unidades de disco estándares, si bien algunas firmas independientes han producido interfaces para disco especiales. Sinclair ha producido el sistema Interface Uno/Microdrive, que utiliza un bucle de cinta que puede almacenar alrededor de 85 K de datos. La cinta está completamente bajo el control del ordenador y cualquier dato individual se puede localizar en cuestión de unos 10 s. Esto representa un rendimiento que está a medio camino entre el de una grabadora de cassette y una unidad de disco, a un precio considerablemente inferior al del sistema de unidad de disco más económico. La Interface Uno ofrece la ventaja de proporcionar una interface RS232 (no estándar) y se puede utilizar para "conexión en red": el enlace en red de hasta 64 Spectrums o máquinas QL.
Un competidor del sistema Interface Uno es la Rotronics Wafadrive. Ésta también emplea bucles de cinta para retener datos, pero incluye interfaces RS232 y Centronics y un programa para tratamiento de textos incluido en su precio. Las cintas se suministran en tres tamaños diferentes (16, 64 y 128 K), obteniéndose la mayor velocidad operativa con la cinta de menor capacidad. En la fotografía vemos el Retronics Wafadrive, la unidad de disco Oric/Atmos, el Torch Disk Pack y la Interface Uno con microdrive de Sinclair



Dispositivos para gráficos

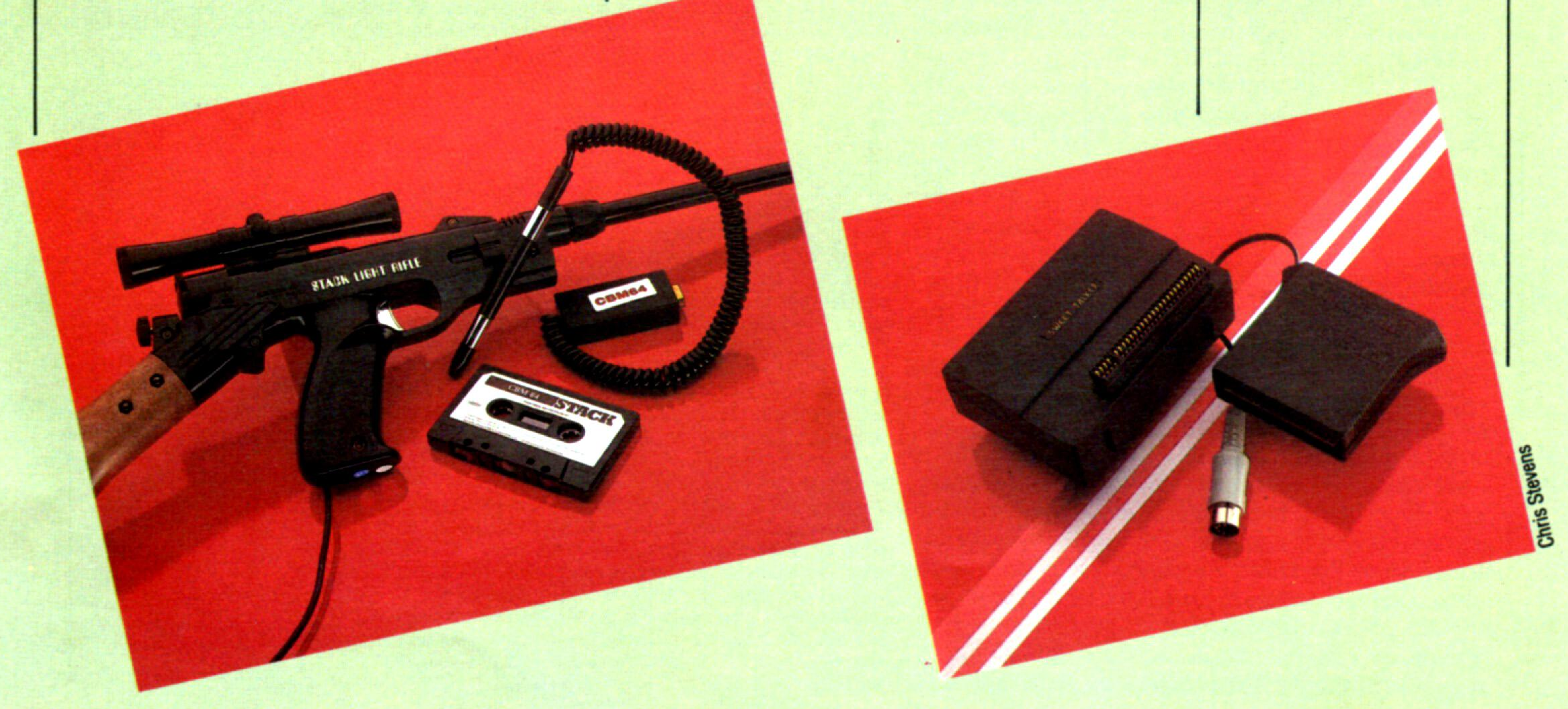
La producción de gráficos con un ordenador personal se simplifica considerablemente si se utiliza alguno de los muchos tipos diferentes de dispositivos para gráficos. Los más económicos son los lápices ópticos, que se pueden emplear para "dibujar" directamente en la pantalla de visualización mediante el empleo de una célula fotoeléctrica para detectar la posición de la punta del lápiz mientras toca la pantalla. Un desarrollo basado en este sistema es el Stack Light Rifle (véanse pp. 710-711).
A muchas personas les resulta difícil dibujar "a mano alzada" en una pantalla de visualización. Seguir líneas sobre una superficie plana es considerablemente más fácil y se fabrican muchos dispositivos que permiten que los usuarios hagan esto. Las tablillas para gráficos utilizan un lápiz especial que le transfiere al ordenador cualquier movimiento efectuado sobre la superficie de la tablilla; ello significa que se las puede utilizar para dibujar imágenes a mano alzada o para calcar imágenes impresas. Otros dispositivos de "dibujo" son los tiralíneas digitales, que hacen uso de resistencias variables situadas en un brazo mecánico para detectar la posición del punzón que llevan fijado en la punta del brazo. Aquí vemos la tablilla Grafpad de British Micro, el tiralíneas Robot Plotter y el lápiz óptico y Light Rifle de Stack.

Modems

El desarrollo de modems económicos para ordenadores personales permite a los usuarios comunicarse entre sí a través de la red telefónica. Para las máquinas equipadas con una interface RS232 estándar se han producido numerosos modems; con el software adecuado, éstos pueden acceder a la base de datos Prestel, que tiene muchas páginas dedicadas enteramente al usuario de ordenador personal. Asimismo, los modems se pueden utilizar para comunicarse con otros usuarios a través de "tablones de anuncios" (bases de datos frecuentemente mantenidas por entusiastas de los micros). Sin embargo, en este campo vuelve a plantearse la cuestión de la compatibilidad: distintos tablones de anuncios utilizan diferentes velocidades de transmisión, y un modem que puede utilizar el Prestel a menudo es inadecuado para comunicaciones con un tablón de anuncios. Ni el Spectrum ni el Commodore 64 poseen interface RS232 incorporada, y por consiguiente no pueden emplear modems estándares. Para el Spectrum, el modem más vendido es el Prism VTX5000. Mediante software en cinta dos Spectrums equipados con modems Prism pueden intercambiar programas o datos. Commodore suministra su propio modem para usar con el 64, y ha creado su propio sistema, Compunet, para conectar en red a los usuarios de Commodore 64.
Los usuarios de un modem deben estar siempre atentos al reloj, dado que los entusiastas pueden encontrarse enseguida con cuantiosas facturas telefónicas. Las tarifas anuales fijas para usuarios de Prestel y el Compunet son igualmente elevadas. La fotografía muestra el Prism VTX5000 y el modem de Commodore.

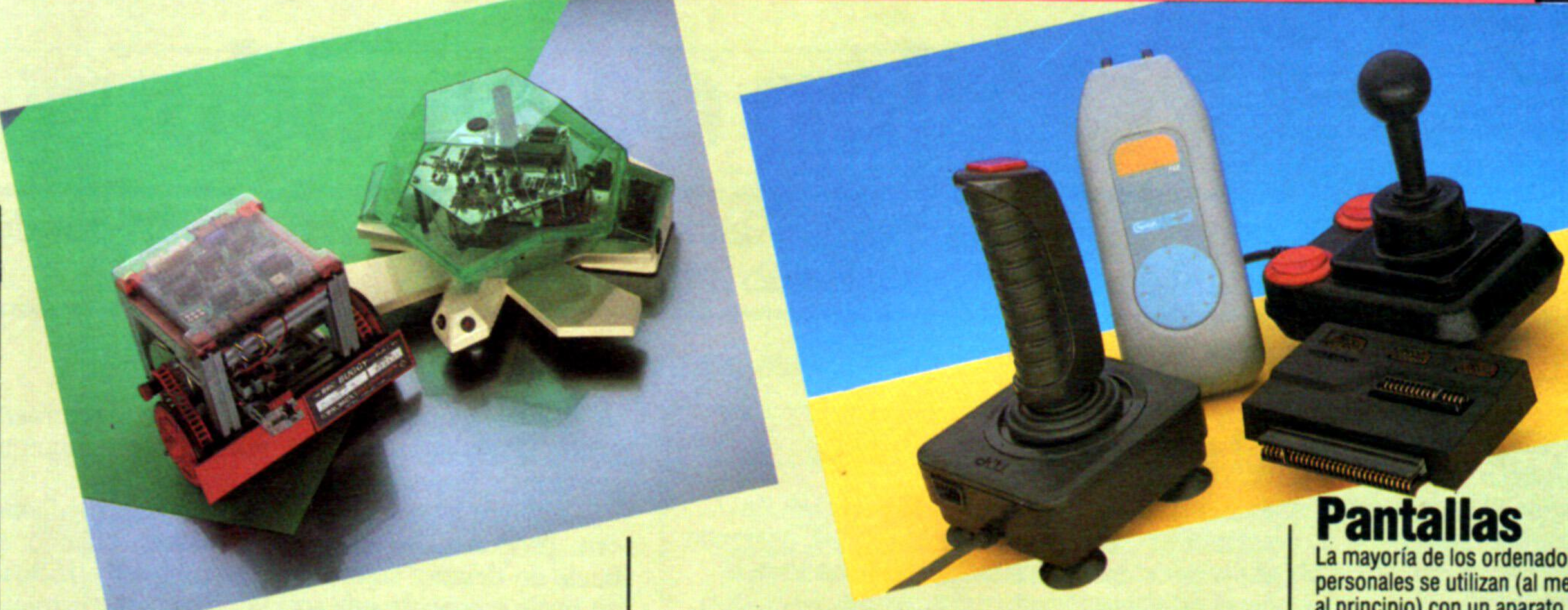
Sintetizadores de voz

Muchas máquinas personales populares pueden producir voz con la adición de una unidad para síntesis de voz. Las unidades existentes se pueden agrupar en dos categorías: una que se suministra con un vocabulario fijo de unas 100 palabras diferentes, y otra que utiliza *alófonos* (conjunto de diferentes sonidos y pausas a partir de los cuales se construyen las palabras).
La gama Currah de unidad de voz emplea el sistema de alófonos, y la firma produce módulos tanto para el Spectrum (Microspeech) como para el Commodore 64 (Speech 64). Algunos juegos para el Spectrum y el Commodore 64, en particular los producidos por Ultimate, poseen voz incorporada; ésta se produce automáticamente si se conecta una unidad Currah. La fotografía muestra el Speech 64 de Currah, y el Sweetalker de Cheetah para el Spectrum.



Chris Stevens





Dispositivos controlados por ordenador

Los ordenadores se pueden utilizar fácilmente para controlar dispositivos del mundo "real". La aplicación que se cita generalmente es el control de un sistema de calefacción central doméstico; ello es bastante factible, aunque casi no merece la pena, dado que tales sistemas suelen llevar incorporado un interruptor de reloj perfectamente eficaz. Mucho más interesantes son los diversos vehículos con ruedas. La Valiant Turtle es un vehículo que se parece algo a una tortuga y que puede producir los gráficos que utiliza el lenguaje LOGO. Este dispositivo se puede acoplar al Spectrum, al Commodore 64 y al BBC Micro, y emplea un haz infrarrojo para comunicarse con el ordenador. El BBC Buggy es un dispositivo de tipo similar.
En la fotografía vemos la Valiant Turtle y el BBC Buggy.

Impresoras-plotter

Para quien utilice un ordenador personal para el desarrollo de programas o para tratamiento de textos, la adquisición de una impresora resulta esencial. La mayoría son matriciales o de rueda margarita. Las primeras utilizan una cuadrícula de pequeños puntos para construir cada letra, permitiendo imprimir gráficos, y son de operación rápida, aunque su calidad de impresión es inferior a la de las de rueda margarita, que son básicamente máquinas de escribir controladas por ordenador. Un sistema alternativo es la pequeña impresora-plotter que se comercializa para los ordenadores Tandy, Atari, Commodore y Oric. Ésta utiliza un papel de unos 10 cm de ancho y está equipada con cuatro pequeños lápices de punta esférica que permiten escribir textos o producir gráficos multicolores. El texto se "dibuja" de la misma forma que los gráficos, y el dispositivo tiene programado un juego completo de caracteres. Otra alternativa es la que representa la impresora térmica Epson P40, que emplea una columna de elementos de calor para "quemar" los caracteres sobre papel especial. Ésta es sumamente económica, a pesar de lo cual proporciona una calidad de impresión razonable y funciona con pilas recargables. También el papel que utiliza es bastante estrecho, pero puede producir una salida impresora de 80 columnas si se emplea la modalidad de impresión compacta. Aquí vemos la impresora-plotter (para Tandy/Radio Shack) y la Epson P40.

Palancas de mando

El primer periférico que suele adquirir el usuario de un ordenador personal es la palanca de mando. Muchos ordenadores están equipados con interfaces adecuadas, y algunas de las máquinas más modernas vienen ya con palancas de mando como estándar. Las palancas más comunes utilizan los conectores "D" de nueve patillas que fueron adoptados en primer lugar por los micros Commodore y Atari, y desde entonces lo han sido por numerosas empresas independientes. Las palancas BBC no son estándares, de modo que en este caso las opciones son más limitadas; en sus micros Plus/4 y 16, la firma Commodore ha ignorado su propio estándar, con lo cual ha obligado a los usuarios a emplear las nuevas palancas diseñadas por la misma.
Recientemente Sinclair ha comercializado la Interface Two (dos), interface para palanca de mando y puerta para cartucho de ROM para el Spectrum. Hasta ese momento no se había proporcionado para esta máquina ninguna interface "oficial" para palanca de mando, y el estándar de *facto* había sido la interface Kempston, cuyas especificaciones han sido adoptadas por muchas otras firmas. Lamentablemente son incompatibles y muchos juegos que son éxitos de ventas funcionarán bastante bien con la interface Kempston, pero no lo harán con la Interface Two, por lo cual ahora Kempston ha producido una que es compatible con el software escrito tanto para la Interface Two como con el antiguo formato Kempston. Una posible alternativa es comprar una interface "programable", que le permite al usuario ejecutar cualquier software, haya sido o no diseñado originalmente para utilizar con palanca de mando.
De las muchas palancas de mando que hay en el mercado, la más inusual es la nueva RAT de Cheetah (véase pp. 1070-1071). Ésta no posee ningún cable que la conecte al ordenador, sino que utiliza un haz infrarrojo para enviar y recibir señales. Por ahora, sólo está a la venta para el Spectrum. El sistema Amstrad es también un tanto inusual. El micro Amstrad posee un único conector para palanca de mando pero se le puede conectar a la primera una segunda palanca.
La fotografía muestra (en el sentido de las agujas del reloj): la palanca Amstrand, la RAT de Cheetah, la Kempston PRO 5000 y la interface Kempston para el ZX Spectrum.

Pantallas

La mayoría de los ordenadores personales se utilizan (al menos al principio) con un aparato de televisión normal como pantalla de visualización. Ello suele plantear problemas, dado que quizá otros miembros de la familia deseen mirar la televisión mientras el usuario del ordenador esté jugando al *Pacman*; de cualquier modo, la calidad de imagen con frecuencia es pobre. La solución es emplear una pantalla, que proporciona una imagen de mejor calidad. El usuario debe asegurarse de que compra el monitor adecuado, puesto que hay dos estándares principales: el RGB y el video compuesto (véase p. 509). Las pantallas de video compuesto se utilizan con micros Atari y Commodore, mientras que BBC, Oric/Atmos y Sinclair QL requieren el formato RGB, que es el que proporciona la mejor calidad de imagen. Algunos micros dependen del altavoz del televisor para producir sonido, de modo que los mismos requerirán pantallas con altavoz incorporado. Varios fabricantes de televisores producen aparatos equipados con interfaces para pantalla; éstos son ideales si el usuario requiere a la vez un televisor y una visualización de gran calidad. Aquí vemos el Microvitec Cub.



Creación de sonido

Ya nos es posible crear señales sonoras a partir del convertidor de digital a analógico diseñado anteriormente

El proyecto

El usuario puede controlar la salida a través de auriculares o bien a través de un estéreo, utilizando el par de conectores de salida que se hallan más cerca del potenciómetro de la caja D/A. Necesitará, asimismo, hacer algunos sencillos cables de conexión. Para auriculares: emplee el enchufe hembra del estéreo, consiga un conector en línea adecuado y suelde los dos extremos positivos a un cable conectado al conector rojo de la caja D/A. Para un estéreo: localice las conexiones audio-IN y tierra y luego haga el cable. Ahora siga estos pasos: conecte la caja buffer y el convertidor D/A y enchufe la caja buffer en la puerta para el usuario; enchufe los cables del amplificador de los auriculares o del estéreo en los conectores D/A; gire el potenciómentro del convertidor D/A totalmente hacia la izquierda y luego suminístrele potencia del transformador a la caja.

Ian McKinnell

```
         LDX #0          2
BUCLE1
         LDA TABLA,X     4
         STA PUERTA      4
         INX             2
         CMP #PASOS      2
         BNE BUCLE1      3
(2 si fracasa el bucle)
```

Después de haber seguido estos pasos, podemos probar el sistema utilizando un corto programa en BASIC. En esencia, el sonido se genera eléctricamente proporcionándole a un micrófono un voltaje oscilante. Podemos generar una salida primaria de voltaje oscilante desde nuestro convertidor D/A cambiando el contenido del registro de datos de la puerta para el usuario de 0 a 255 y al revés en rápida sucesión. Entre el siguiente programa y ejecútelo. Gire el potenciómetro D/A en el sentido de las agujas del reloj hasta que se escuche sonido.

```
10 REM .... GENERADOR DE SONIDO BASIC CBM ....
20 RDD=56579:REGDAT=56577
30 POKERDD,255
35 N=1
40 POKE REGDAT,0:FORI=1TON:NEXT:POKE REGDAT,255:GOTO40
```

```
10 REM .... GENERADOR DE SONIDO BASIC BBC ....
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=255
35 N=1
40 ?REGDAT=0:FOR I=1TON:NEXT:? REGDAT=255:GOTO40
```

Observe que el programa en BASIC tiene una estructura repetitiva, concentrada toda en una única línea para conseguir la máxima velocidad. Hay un bucle de demora insertado entre los puntos en que el registro de datos se establece en 255 y su establecimiento en 0. El valor N de la línea 35 establece la longitud de esta demora. Pruebe de alterar el valor de N y ejecutar de nuevo el programa. Observará que la altura del tono oído baja mientras aumenta el valor de N.

La mayor altura que se puede obtener mediante este programa en BASIC se producirá cuando el bucle de demora se elimine por completo. Incluso un bucle ejecutado una sola vez tiene un efecto audible sobre la altura de la nota escuchada.

Si ha experimentado con distintos valores de N en el programa en BASIC que hemos ofrecido, habrá notado que el cambio del valor de N en 1 tiene un efecto significativo sobre la altura de la nota. El BASIC no es lo suficientemente rápido como para permitirnos controlar con precisión la velocidad de oscilación. Para eso debemos utilizar el lenguaje máquina.

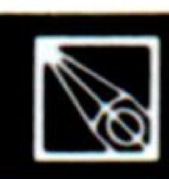
En el próximo capítulo de *Bricolaje* estudiaremos el difícil problema de controlar altura y volumen mediante código máquina. Aquí nos concentraremos en desarrollar un programa para producir distintas formas de onda. La forma de onda producida por el programa en BASIC utilizado anteriormente era una forma de onda cuadrada. No obstante, es posible producir otras formas de onda, que alteran la calidad del sonido oído. Podemos sintetizar digitalmente ondas senoidales y aserradas tomando un cierto número de muestras de la forma de onda y colocándolas en una tabla. El programa que se requiere en lenguaje máquina para colocar estas muestras una después de la otra en el registro de datos es esencialmente muy simple. Nuestra ilustración muestra estas tres formas de onda, acompañadas de las tablas para las formas de onda senoidal y aserrada. Se divide en pasos el ciclo de la forma de onda y se muestrean estos pasos; a la izquierda detallamos el bucle del programa que envía estas muestras a través de la puerta para el usuario.

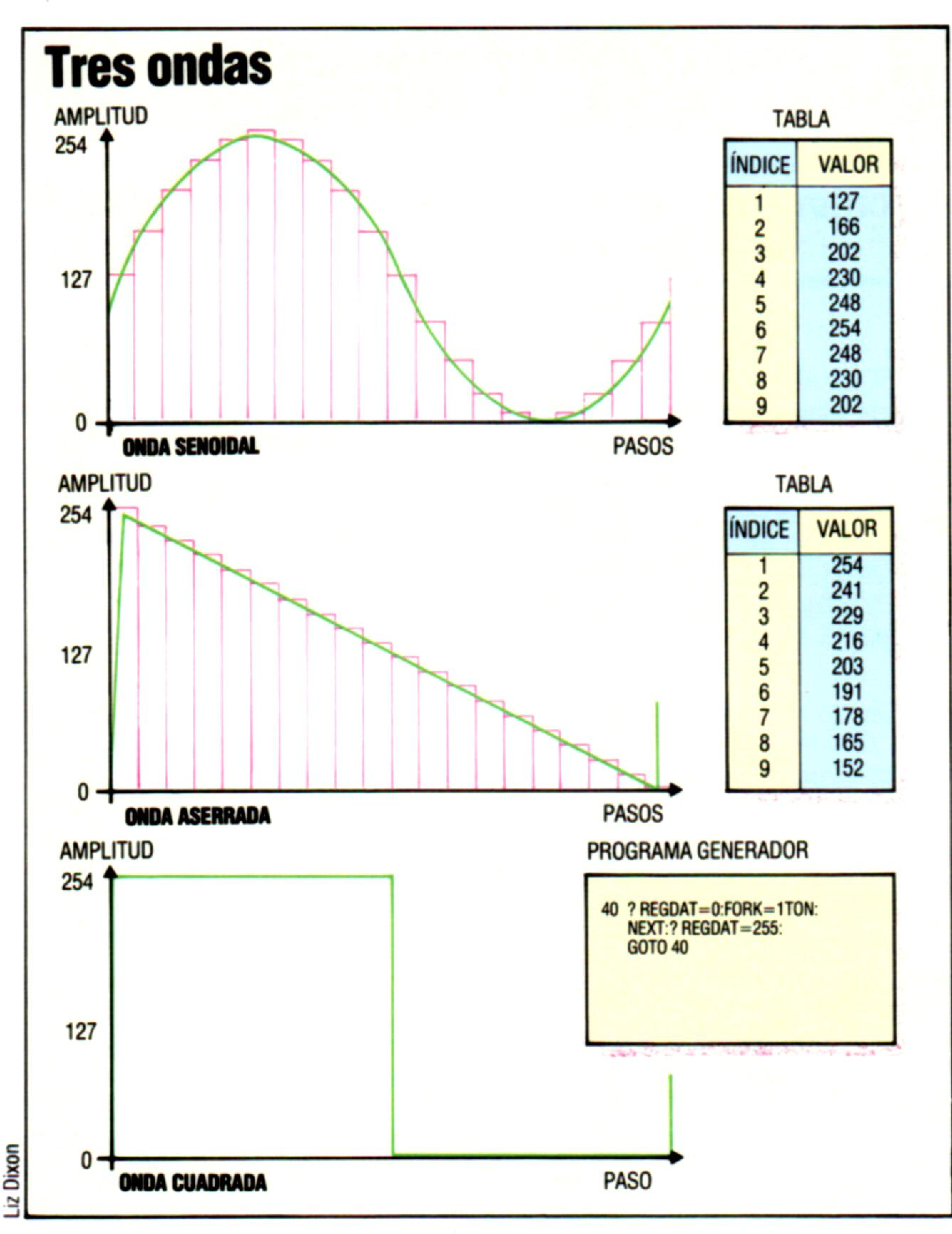
Cuando se trata de producir sonido, el temporizado es esencial. A continuación de cada instrucción está la cantidad de ciclos de máquina requeridos para ejecutar esta instrucción. A partir de esta fórmula podemos calcular el número total de ciclos de máquina que lleva producir un ciclo completo de forma de onda: número de ciclos de máquina $= 2 + (4 + 4 + 2 + 2 + 3) \times \text{pasos} - 1 = 1 + 15 \times \text{pasos}$.

Si la onda se divide en 80 muestras, entonces el número de ciclos de máquina requeridos para producir un ciclo de forma de onda es 1 201.

Dado que en código 6502 cada ciclo de máquina toma alrededor de una millonésima de segundo, el número total de ciclos de forma de onda que se puede producir en un segundo (es decir, la frecuencia de la nota) viene dado por este cálculo: frecuencia = 1 000 000/1201 = 832 Hz. Puesto que do central es 512 Hz, la nota producida tendrá una altura de unas pocas notas más arriba de do central.

Se puede ver que el número de pasos de muestra





ÍNDICE	VALOR
1	127
2	166
3	202
4	230
5	248
6	254
7	248
8	230
9	202

ÍNDICE	VALOR
1	254
2	241
3	229
4	216
5	203
6	191
7	178
8	165
9	152

Creación de formas
Las formas de onda senoidal y aserrada se crean decidiendo primero cuántos pasos de muestra han de constituir un ciclo de la onda. Estas muestras de la amplitud de la onda se calculan luego y se almacenan en una tabla. Los valores de esta tabla se pueden, entonces, copiar en secuencia en el registro de datos de la puerta para el usuario y, por tanto, en el convertidor de digital a analógico, donde se convierten en niveles de voltaje. La ventaja de la tabla es que permite realizar de antemano los cálculos que tanto tiempo llevan; la verdadera generación de la forma de onda, por consiguiente, lleva poco tiempo, y ello hace posible una gama de frecuencias de varias octavas. Sin el uso de la tabla la gama se limitaría a dos octavas.
La onda cuadrada se puede generar mediante un programa en BASIC debido a que el proceso es muy sencillo. La lentitud de este lenguaje, sin embargo, limita considerablemente la gama de frecuencias

por el cual hemos elegido dividir nuestra forma de onda tiene un efecto directo sobre la frecuencia de la nota final. Duplicando la cantidad de pasos de muestra dividiremos por la mitad la frecuencia de ésta. Obviamente, cuantas más muestras tomemos de la nota, más probabilidades tendremos de acercarnos a obtener la calidad de nota que estamos sintetizando, pero esto siempre se debe comparar con la frecuencia máxima final que se puede obtener para un número dado de pasos.

Es probable que un ciclo de forma de onda no tenga la longitud suficiente como para ser audible, por lo que debemos también incluir código para repetir la sección de código generadora de la forma de onda un número de veces establecido. La cantidad de repeticiones se puede determinar estableciendo un valor de contador y disminuyéndolo hasta cero. Para obtener una amplia gama de valores de contador, se ha utilizado un número de 16 bits almacenado en dos posiciones adyacentes. Además de este código, las interrupciones se desactivan al principio del programa mediante SE1 y se vuelven a activar mediante CL1 al final. Si se produjeran interrupciones durante la ejecución, ello restaría precisión al temporizado del programa. Sin embargo, no es posible desactivar todas las interrupciones; las interrupciones no enmascarables, si se producen durante la ejecución del programa, pueden causar algunos errores de temporizado.

Los datos de la forma de onda se deben colocar en la memoria dentro de una tabla, con cada tipo de forma de onda ocupando 80 posiciones consecutivas. En la versión Commodore, la tabla de la onda senoidal está ubicada en la memoria a partir de $C000; la tabla de la onda aserrada está en $C050 y la tabla de la onda cuadrada empieza en $C0A0. El programa en lenguaje máquina está diseñado para recurrir por defecto a cargar datos desde la tabla de onda senoidal utilizando el direccionamiento indexado, pero podemos pasar a otra tabla modificando el programa directamente con un POKE desde BASIC. La parte LDA de LDA SINE,X está en la posición $C103. La dirección de comienzo de la tabla de datos a cargar tiene su byte *lo* en la posición $C104 y su byte *hi* en la posición $C105. Para modificar la dirección de comienzo de los datos a cargar lo único que hay que hacer es cambiar el número retenido en $C104. Corrientemente, para una onda senoidal, esta posición contendrá 0; si deseamos cambiar a una onda aserrada, entonces deberemos cambiar el contenido de $C104 por 80. Cambiando el contenido de esta posición a 160 cambiaremos la forma de onda a una onda cuadrada.

La versión BBC también está diseñada de modo que las tablas empiezan al principio de una nueva página de la memoria. Por esta razón el byte *hi* de todas las direcciones de comienzo de las tablas es el mismo, y sólo hemos de modificar el byte *lo*. En un BBC Modelo B normalmente configurado en modalidad 7, HIMEM es &7C00. Bajando tres páginas el tope de la memoria destinamos un espacio más que suficiente para las tablas de referencia y el programa en lenguaje máquina.

Para el Commodore

```
                ;**************************
                ;**************************
                ;**                      **
                ;**      GENERADOR       **
                ;**   DE SONIDO CBM 64   **
                ;**                      **
                ;**************************
                ;**************************
                PUERTA   =56577               ;DIRECCION REGISTRO DATOS
                PASOS    =80                  ;NUMERO DE PASOS EN ONDA
                *=$C000
                ;
                ;****PREPARAR ZONA TABLA DE DATOS****
                SENO     *=*+STEPS
                ASERR    *=*+STEPS
                CUADR    *=*+STEPS
                NUM      *=*+2
                CONT     *=*+2
                ;
                ;****PROGRAMA PRINCIPAL****
                ;
78                       SEI
AD F0 C0                 LDA NUM
8D F2 C0                 STA CONT             ;ESTABLECER VALOR CONTADOR
AD F1 C0                 LDA NUM+1
8D F3 C0                 STA CONT+1
                ;
                LOOP2
A2 00                    LDX #$00
                LOOP1
BD 00 C0                 LDA SENO,X           ;TOMAR DATOS
8D 01 DD                 STA PUERTA           ;PONER EN PUERTA USUARIO
E8                       INX
E0 50                    CPX #PASOS
D0 F5                    BNE LOOP1            ;FIN DE UN CICLO
                ;
                ;****DECREMENTAR CONTADOR****
AD F2 C0                 LDA CONT
38                       SEC
E9 01                    SBC #$01
8D F2 C0                 STA CONT
AD F3 C0                 LDA CONT+1
E9 00                    SBC #$00
8D F3 C0                 STA CONT+1
D0 E0                    BNE LOOP2            ;SI HIBYTE>0
A9 00                    LDA #$00
CD F2 C0                 CMP CONT
D0 D9                    BNE LOOP2            ;SI LOBYTE>0
58                       CLI
60                       RTS
```

El código máquina se puede entrar en su ordenador entrando el listado fuente proporcionado y ensamblándolo para crear un archivo hexa que se pueda cargar siempre que así se requiera. Las tablas se pueden generar ejecutando el siguiente programa:

```
900 REM•••PROGRAMA LLAMADA DE SONIDO CBM•••
910 :
915 DN=8:REM PARA CASSETTE DN=1
920 IF A=0 THEN=1:LOAD"5SOUND.HEX",DN,1
999 :
1000 REM••••PREPARAR VALORES DATOS••••
1005 S=80          : REM NUMERO DE PASOS
1007 TB=12•4096    :REM COMIENZO DE ZONA DATOS
1008 :
1010 REM••ONDA SENOIDAL••
1020 FOR I=0 TO S-1
1030 Y=127•SIN(X)+127
1040 POKE TB+I,Y
1045 X=X+2/S
1050 NEXT I
1060 :
1065 REM••••ONDA ASERRADA••••
1070 Y=255:TB=TB+S
1080 FOR I=0 TO S-1
1090 POKE TB+I,Y
1100 Y=Y-255/S
1110 NEXT I
1120 :
1125 REM••••ONDA CUADRADA••••
1130 Y=255:TB=TB+S
1140 FOR I=0 TO S/2-1
1150 POKE TB+I,Y
1160 NEXT I
1165 Y=0
1170 FOR I=S/2 TO S-1
1180 POKE TB+I,Y
1190 NEXT I
1999 :
2000 REM••••VISUALIZAR TABLAS DATOS••••
2005 TB=12•4096
2010 FOR I=TB TO TB+3•S-1
2020 PRINTI,I-TB,PEEK(I)
2030 NEXT
```

Después de ejecutar este programa, digite NEW y después entre este programa muestra que ilustra cómo utilizar el código máquina, dando las direcciones SYS y POKE requeridas para interactuar con el código máquina desde BASIC. Este programa le pide al usuario que entre el tipo de onda requerido y luego produce un tono cada vez que se pulsa una tecla.

```
10 REM •••• SONIDO CBM 64 ••••
20 REM •••• PROGRAMA MUESTRA ••••
30 :
40 RDD=56579:POKE RDD,255: REM TODAS SALIDA
65 CL=49392 :REM POSICION BYTE LO CONTADOR
67 TL=49412 :REM DIGITAR POSITION BYTE LO
70 SONIDO=49396 :REM DIRECCION DE COMIENZO PROGRAMA
75 REM •• ESTABLECER VALOR CONTADOR ••
80 NUM=80:NHI=INT(NUM/256):NLO=NUM-256•NHI
82 POKE CL,NLO:POKE CL+1,NHI
83 :
85 PRINTCHR$(147):REM BORRAR PANTALLA
86 INPUT"TIPO DE ONDA (0)SENOIDAL (1)ASERRADA (2)CUADRADA";WT
87 POKE TL, WT•S
88 PRINT:PRINT"PULSE CUALQUIER TECLA (RUN/STOP PARA TERMINAR)"
90 GETA$:IF A$=""THEN90:REM ESPERAR UNA TECLA
100 SYS SOUND:REM LLAMAR CODIGO MAQUINA
110 IF A$="X" THEN 85
120 GOTO 90
```

Si no posee ensamblador o no entiende el lenguaje assembly aun así puede utilizar el programa en código máquina digitando este cargador en BASIC y ejecutándolo. En este caso, puede omitir la línea 920 del programa que prepara la tabla.

```
10 REM •••• CARGADOR BASIC PARA SONIDO CBM ••••
20 REM •••• CODIGO MAQUINA ••••
30 FOR I=49396 TO 49449
40 READ A:POKE I,A
50 CC=CC+A
60 NEXT I
70 READ CS:IF CC<>CS THEN PRINT"ERROR SUMA CONTROL":END
100 DATA120,173,240,192,141,242,192
110 DATA173,241,192,141,243,192,162,0
120 DATA189,0,192,141,1,221,232,224,80
130 DATA208,245,173,242,192,56,233,1
140 DATA141,242,192,173,243,192,233,0
150 DATA141,243,192,208,224,169,0,205
160 DATA242,192,208,217,88,96
170 DATA9115:REM•SUMA DE CONTROL•
```

Para el BBC

En el BBC, el proceso de combinar BASIC y código máquina es más fácil que en el caso del C64.

```
5 REM •••• PROGRAMA DE SONIDO BBC ••••
8 MODE 7
10 HIME=HIMEM-&0301
20 MC%=HIMEM+1
30 RDD=&FE62:?RDD=255:REM TODAS SALIDA
40 puerta=&FE60:REM REG DATOS FUERA USUARIO
50 pasos=80     :REM NUMERO DE PASOS POR ONDA
60 comienzo_tabla=MC%
70 PROCpreparar_tablas
80 PROCcodigo_maquina
90 PROCprograma_muestras
999 END
1000 DEF PROCcodigo_maquina
1010 :
1020 FOR opt%=1 TO 3 STEP 3
1030 P%=MC%
1060 seno=P%:       P%=P%+pasos
1070 aserr=P%:      P%=P%+pasos
1080 cuadr=P%:      P%=P%+pasos
1090 num=P%:        P%=P%+2
1100 cont=P%:       P%=P%+2
1110 [
1120 OPT opt%
1130 \•••• AQUI COMIENZA EL PROGRAMA PRINCIPAL ••••
1150 .sonido
1160          SEI
1170          LDA num
1180          STA cont
1190          LDA num+1
1200          STA cont+1
1220 .bucle2
1230          LDX #&00
1240 .bucle1
1250          LDA seno,X
1260          STA puerta
1270          INX
1280          CPX #pasos
1290          BNE bucle1
1310 \•••• DECREMENTAR CONTADOR ••••
1320 \
1330          LDA cont
1340          SEC
1350          SBC #&01
1360          STA cont
1370          LDA cont+1
1380          SBC #&00
1390          STA cont+1
1400          BNE bucle2
1410          LDA #&00
1420          CMP cont
1430          BNE bucle2
1440          CLI
1450          RTS
1455 ]
1460 NEXT opt%
1480 ENDPROC
2000 DEF PRCOpreparar_tablas
2020 REM •••• ONDA SENOIDAL ••••
2025 x=0
2030 FOR I=0 TO pasos-1
2040 y=127•SIN(x)+127
2050 ?(comienzo_tabla+1)=y
2060 x=x+2•PI/pasos
2070 NEXT I
2090 REM •••• ONDA ASERRADA ••••
2100 y=255:comienzo_tabla=comienzo_tabla+pasos
2110 FOR I+0 TO pasos-1
2120 ?(comienzo_tabla+I)+y
2130 y=y-255/pasos
2140 NEXT I
2160 REM •••• ONDA CUADRADA ••••
2170 y=255:comienzo_tabla=comienzo_tabla+pasos
2180 FOR I=0 TO pasos/2-1
2190 ?(comienzo_tabla+1)=y
2200 NEXT I
2220 y=0
2230 FOR I=pasos/2 TO pasos-1
2240 ?(comienzo_tabla+I)=y
2250 NEXT I
2270 REM •••• VISUALIZAR TABLAS DE DATOS ••••
2280 comienzo_tabla=MC%
2290 FOR I=comienzo_tabla TO comienzo_tabla+3•pasos-1
2300 PRINT ˜I, ˜(I-comienzo_tabla),? I
2310 NEXT I
2330 ENDPROC
3000 DEF PROCprograma_muestra
3020 cont=MC%+3•pasos:REM POSICION BYTE LO CONTADOR
3030 tipo=bucle1+1:REM POSICION BYTE LO TIPO
3040 valor_contador=80
3050 hi_contador=valor_contador DIV 256
3060 lo_contador=valor_contador MOD 256
3070 ?cont=lo_contador
3080 cont?1=hi_contador
3090 CLS
3100 INPUT"TIPO DE ONDA (0) SENOIDAL (1) ASERRADA (2) CUADRADA";onda
3110 ?tipo=onda•pasos
3120 REPEAT
3125 PRINT"PULSE CUALQUIER TECLA (X PARA SALIR)"
3130 A$=GET$
3140 CALL sonido
3150 UNTIL A$="X"
3160 GOTO 3090
```



Manejando cifras

En este capítulo de nuestro curso analizaremos las facilidades que ofrece el LOGO para trabajar con números

Casi todas las implementaciones del LOGO soportan tanto aritmética de enteros como de reales (decimales), utilizando los operadores infijos + – * /. Estos operadores se denominan infijos porque se escriben entre los números con los cuales trabajan; por ejemplo, 3+4. Algunas versiones de LOGO incluyen también aritmética de "prefijos", según la cual nuestro ejemplo se escribiría SUM 3 4. Una ventaja de esta notación es que es coherente con la forma en que se escriben las otras operaciones e instrucciones del LOGO.

El LOGO MIT sólo soporta aritmética de infijos, pero es sencillo programar formas con prefijos si así se requiere. Defina SUMA y PRODUCTO y pruébelos:

```
TO SUMA :A :B
  OUTPUT :A + :B
END

TO PRODUCTO :A :B
  OUTPUT :A * :B
END
```

La "precedencia" de operaciones (el orden por el cual se llevan a cabo) sigue las reglas matemáticas habituales. Todo lo que esté entre paréntesis se realiza primero, seguido de multiplicaciones y divisiones, y finalmente sumas y restas:

```
PRINT (3 + 4) * 5
PRINT 3 + 4 * 5
```

Ahora pruebe las formas de prefijo:

```
PRINT PRODUCTO 5 SUMA 3 4
PRINT SUMA 3 PRODUCTO 4 5
```

Esto demuestra otra ventaja de las formas de prefijo: no existe necesidad alguna de reglas de precedencia y la línea se evalúa de la misma manera que cualquier otra línea de instrucciones del LOGO.

La habitual operación de la división (/) da el resultado como un número real. Otras dos operaciones, QUOTIENT (cociente) y REMAINDER (resto), suelen ser útiles para trabajar con enteros.

```
QUOTIENT 47 5 es 9
REMAINDER 47 5 es 2
```

Un método estándar para convertir un número en base 10 a binario es ir dividiendo el número por dos hasta que el resultado sea cero. El número binario resultante se halla escribiendo los restos de cada etapa por orden inverso. Por ejemplo, para convertir 12 a binario:

```
12/2 = 6;resto = 0
 6/2 = 3;resto = 0
 3/2 = 1;resto = 1
 1/2 = 0;resto = 1
```

De este modo, leyendo los restos hacia arriba, hallamos que el decimal 12 es 1100 en binario.

Utilizando QUOTIENT y REMAINDER podemos implementar esta técnica en LOGO fácilmente. Colocando la sentencia de impresión *después* de la llamada recursiva obtenemos los restos impresos en el orden correcto (inverso).

```
TO BIN :X
  IF :X=0 THEN STOP
  BIN QUOTIENT :X 2
  PRINT1 REMAINDER :X 2
END
```

Existen dos operaciones para redondear números: INTEGER y ROUND. INTEGER produce la parte entera de un número, simplemente ignorando cualquier cifra que haya después del punto decimal, y ROUND redondea un número aumentándolo o disminuyéndolo hasta el número entero más cercano.

Los siguientes procedimientos calculan el interés compuesto sobre una inversión a un porcentaje dado de interés. En IMPRESION.EXCELENTE, INTEGER se utiliza para obtener las pesetas, y ROUND, para redondear los céntimos al número entero más cercano.

```
TO COMPUESTO :PRINCIPAL :PRCNT :AÑOS
  IF :AÑOS=0 THEN IMPRESION.EXCELENTE
  :PRINCIPAL STOP
  COMPUESTO :PRINCIPAL*(1+ :PRCNT/100)
    :PRCNT :AÑOS – 1
END

TO IMPRESION.EXCELENTE :DINERO
  MAKE"PESETAS INTEGER :DINERO
  MAKE"CENTIMOS ROUND (:DINERO –
  :PESETAS)*100
  (PRINT :PESETAS"PESETAS :CENTIMOS
    "CENTIMOS)
END
```

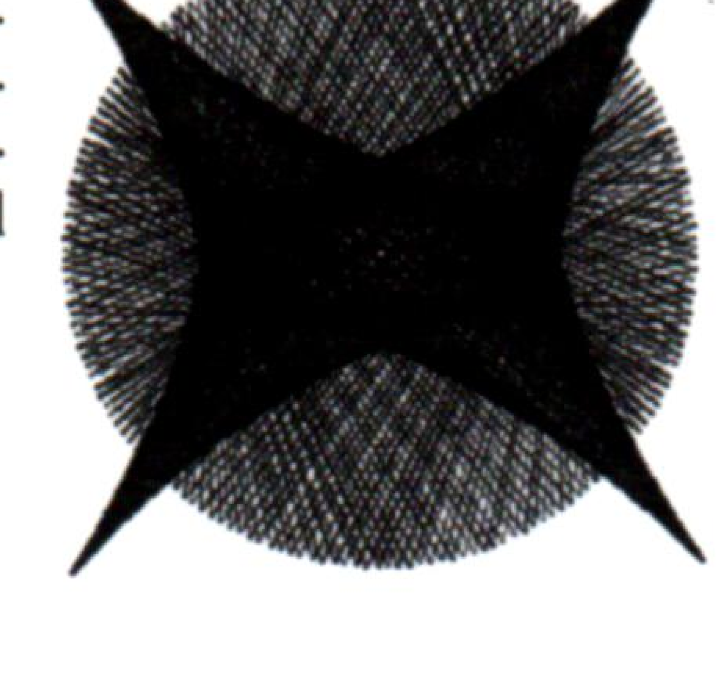
Hora de probar

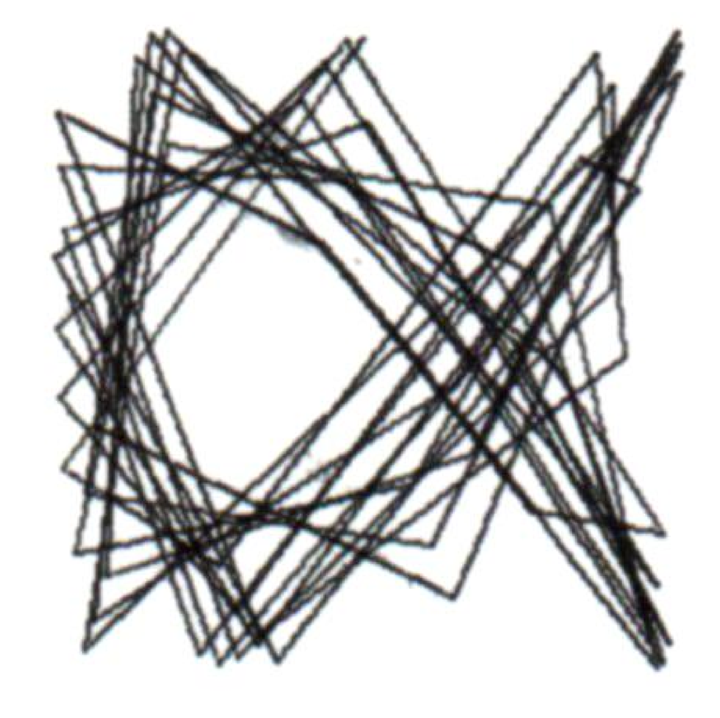
Anteriormente ya hemos utilizado =, < y > como operadores lógicos en comparaciones en numerosos procedimientos. Se pueden emplear las operaciones lógicas ALLOF, ANYOF y NOT para combinar otras condiciones. ALLOF es verdadera si sus dos entradas son verdaderas, ANYOF es verdadera si alguna de sus entradas es verdadera, y NOT es verdadera si su entrada es falsa. De modo que tenemos:

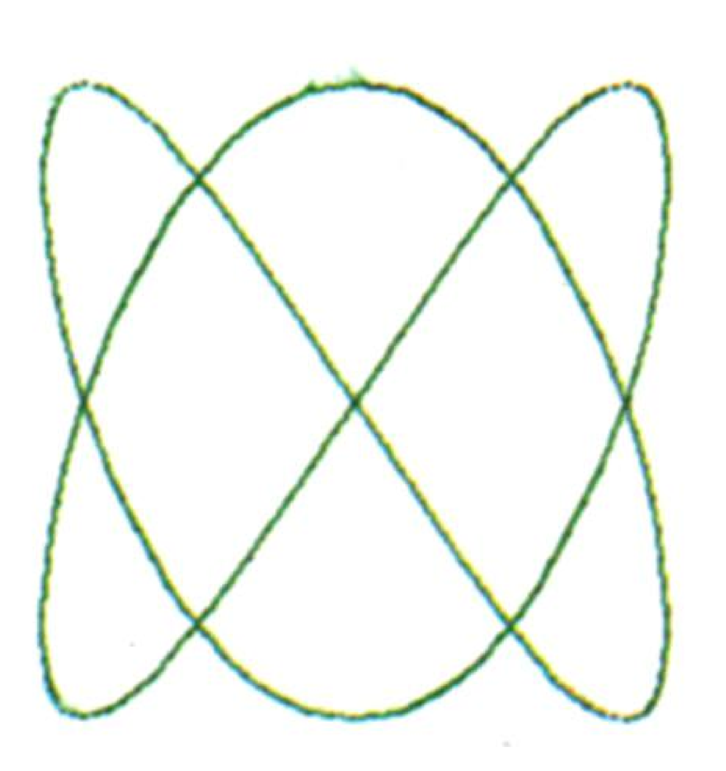
```
IF ANYOF :X > 0 :X=0 THEN PRINT"POSITIVO
IF NOT :X < 0 THEN PRINT "POSITIVO
IF ALLOF :X > 0 :X <100 THEN PRINT
  [ENTRE 0 Y 100]
```

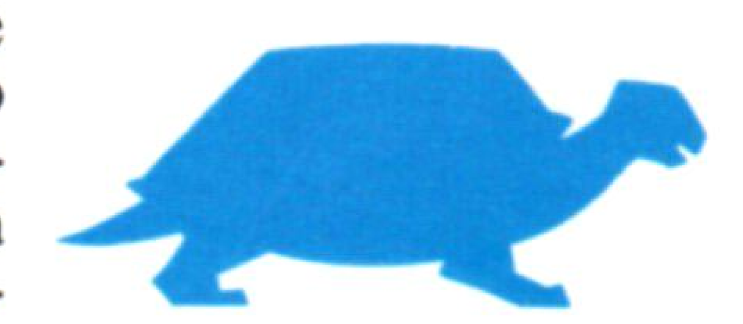
La operación NUMBER? produce TRUE (verdadero) si la entrada es un número, de lo contrario devuelve FALSE (falso). La utilizamos en el procedimiento PRIMO?, que produce TRUE si su entrada es un número primo, o, de lo contrario, FALSE. Empieza por verificar que la entrada sea realmente un núme-

FIGURAS LISSAJOUS

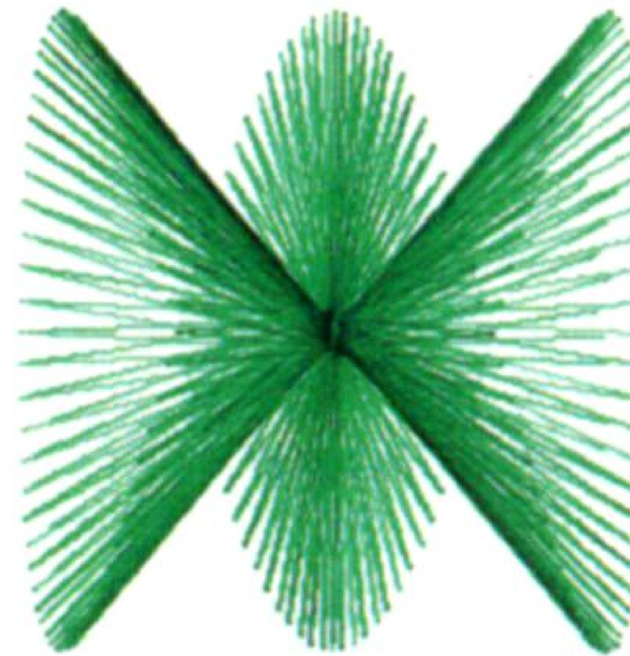


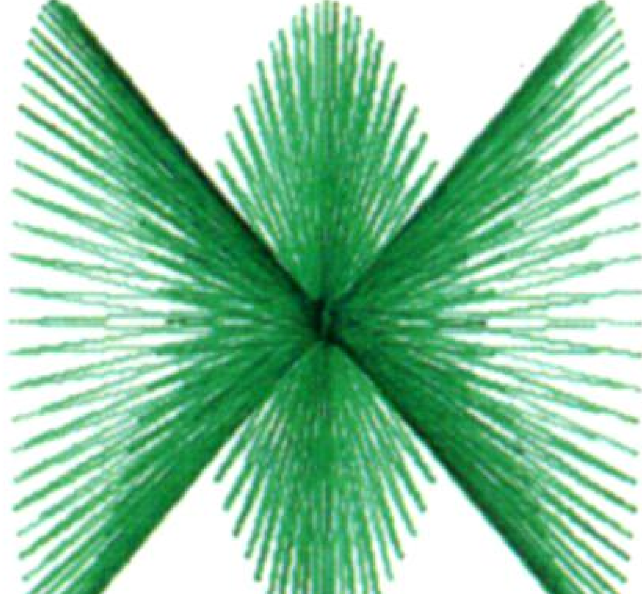


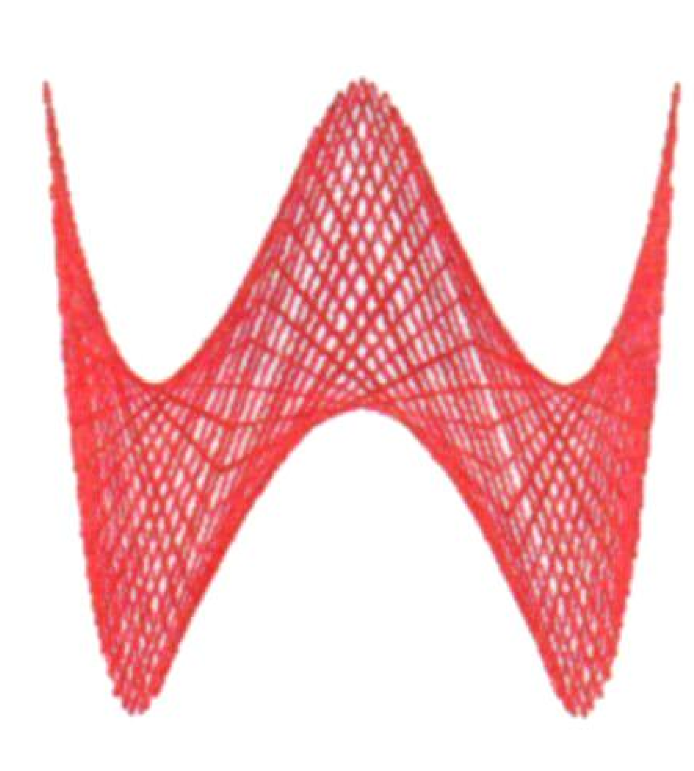




FIGURAS LISSAJOUS







ro y que sea mayor que dos. PRUEBA.PRIMO verifica entonces si hay algún entero entre la raíz cuadrada del número y dos que sea divisible por el mismo, sin que quede ningún resto.

```
TO PRIMO? :NUM
   IF NOT NUMBER? :NUM THEN PRINT [NO ES
      UN NUMERO] STOP
   IF :NUM < 2 THEN OUTPUT"FALSE
   OUTPUT PRUEBA.PRIMO :NUM INTEGER SQRT
      :NUM
END

TO PRUEBA.PRIMO :NUM :HECHO
   IF :HECHO=1 THEN OUTPUT"TRUE
   IF (REMAINDER :NUM :HECHO)=0 THEN OUTPUT
      "FALSE
   OUTPUT PRUEBA.PRIMO :NUM :HECHO - 1
END
```

Números aleatorios

RANDOM n produce un entero al azar entre 0 y n−1. El procedimiento BORRACHA hace que la tortuga trastabillee a través de la pantalla, girando en un ángulo al azar a cada paso. La entrada A da el tamaño máximo del giro que se puede describir en cualquier momento. Si ejecuta este procedimiento comprobará que la tortuga gira en círculos imprecisos, moviéndose hacia la izquierda o la derecha según el valor que se le haya asignado a A.

```
TO BORRACHA :A
   FORWARD 1
   RIGHT (-:A/2+RANDOM :A)
   BORRACHA :A
END
```

El llamado *método Monte Carlo* es una técnica para resolver problemas matemáticos mediante el empleo de números aleatorios.

Caminando sobre la línea

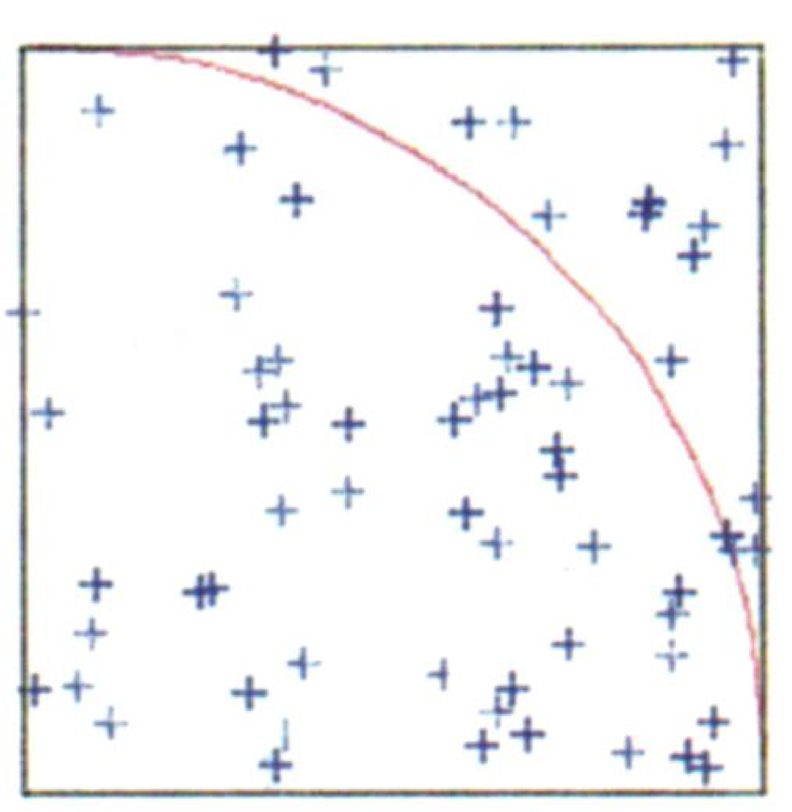
El teorema del "recorrido del borracho" postula que al cabo de N pasos en direcciones totalmente aleatorias, la probabilidad de que la distancia del borracho respecto al punto de origen sea inferior a la raíz cuadrada de N pasos es mayor de 0,5. Ésta es una predicción estadística basada en una gran cantidad de pasos; el LOGO le ofrece la posibilidad de probarla:

```
TO BORRACHO :NUM PASOS:
   :PASO
CS REPEAT :NUM PASOS [RT
   (RANDOM 361) FD
   :PASO]
END
```





RECORRIDO DEL BORRACHO



"PI" LLEGA A MONTE CARLO

En nuestra demostración hallaremos una aproximación a *pi* empleando este método. La ilustración muestra un cuarto de círculo dibujado dentro de un cuadrado. La superficie del cuadrado es de 100 × 100 unidades cuadradas, y la superficie del cuarto de círculo es (1 ÷ 4) × *pi* × 100 × 100 unidades cuadradas. La razón de las superficies círculo ÷ cuadrado es igual a *pi* ÷ 4. Ahora clave un alfiler al azar sobre el cuadrado 1 000 veces y cuente cuántas veces el alfiler ha caído dentro del cuarto de círculo; a este número llámelo IN. El valor de IN/1000 debe ser aproximadamente el mismo que el resultado de: círculo ÷ cuadrado, es decir, *pi* ÷ 4. De modo que si hacemos el experimento, multiplicamos IN por cuatro y dividimos por 1 000, el resultado debería ser una aproximación a *pi*. Eso es precisamente lo que hacen los siguientes procedimientos:

```
TO MC
   DRAW
   PU
   MAKE"IN 0
   MC1 1000 100 100
   (PRINT [EL VALOR DE PI ES]0.004*(:IN))
END

TO MC1 :NUM :XNUM :YNUM
   IF :NUM = 0 THEN STOP
   PUNTO.ALEATORIO :XNUM :YNUM
   IF DENTRO? THEN MAKE "IN :IN + 1
   MC1 :NUM-1 :XNUM :YNUM
END
```

El procedimiento MC simplemente establece las condiciones, llama a MC1 e imprime los resultados. MC1 realiza la mayor parte del trabajo, llama a PUNTO.ALEATORIO para posicionar la tortuga y luego incrementa IN si el punto está dentro del círculo. Esto continúa hasta que el procedimiento se haya llevado a cabo la cantidad correcta de veces.

```
TO PUNTO.ALEATORIO :XNUM :YNUM
   SETXY RANDOM :XNUM RANDOM :YNUM
END

TO DENTRO?
   IF (XCOR*XCOR+YCOR*YCOR) < 10000
      THEN OUTPUT "TRUE
   OUTPUT "FALSE
END
```

PUNTO.ALEATORIO coloca a la tortuga dentro del cuadrado en una posición al azar, mientras que DENTRO? verifica si la tortuga está dentro del círculo. Ejecutar esto llevará bastante tiempo, pero finalmente se obtendrá un valor para *pi* de 3,15999.

Las curvas Lissajous son una interesante familia de curvas en las cuales la coordenada *x* de cada

Steve Malone



punto está determinada por la función seno y la coordenada *y* por la función coseno:

```
TO LJ :COEF1 :COEF2 :PASO
   DRAW PU HT
   POS :COEF1 :COEF2 0 PD
   LJ1 :COEF1 :COEF2 0 :PASO
END
TO POS :COEF1 :COEF2 :ANGULO
   MAKE "X 100*SIN (:COEF1* :ANGULO)
   MAKE "Y 100*COS (:COEF2* :ANGULO)
   SETXY :X :Y
END
TO LJ1 :COEF1 :COEF2 :ANGULO :PASO
   POS :COEF1 :COEF2 :ANGULO
   LJ1 :COEF1 :COEF2 (:ANGULO + :PASO) :PASO
END
```

Complementos al LOGO

Las versiones LCSI incluyen aritmética de prefijo. El LOGO Atari posee SUM y PRODUCT; el LOGO Spectrum posee también DIV, y el LOGO Apple tiene QUOTIENT, que corresponden ambas a QUOTIENT del LOGO MIT.
Se utiliza INT en lugar de INTEGER.
En vez de NUMBER? se emplea NUMBERP.
Los operadores lógicos poseen los nombres más usuales de AND, OR y NOT.
IF tiene una sintaxis distinta: en lugar de PRINT1 se utiliza IF :X = 0 PRINT "TIPO CERO.
En lugar de SETXY se utiliza SETPOS (seguido de una lista). Utilice CS en lugar de DRAW.

Ejercicios de LOGO

1. Escriba un procedimiento para producir la enésima potencia de un número, de modo que POTENCIA 4 2 produzca 16.
2. Escriba un conjunto de procedimientos para convertir un número decimal a hexadecimal.
3. Escriba un procedimiento PAR? que produzca TRUE si un número es par y FALSE si no lo es.
4. Utilice el método Monte Carlo para hallar la superficie existente bajo la curva $y=x^2$ entre $x=0$ y $x=10$.

Respuestas a los ejercicios

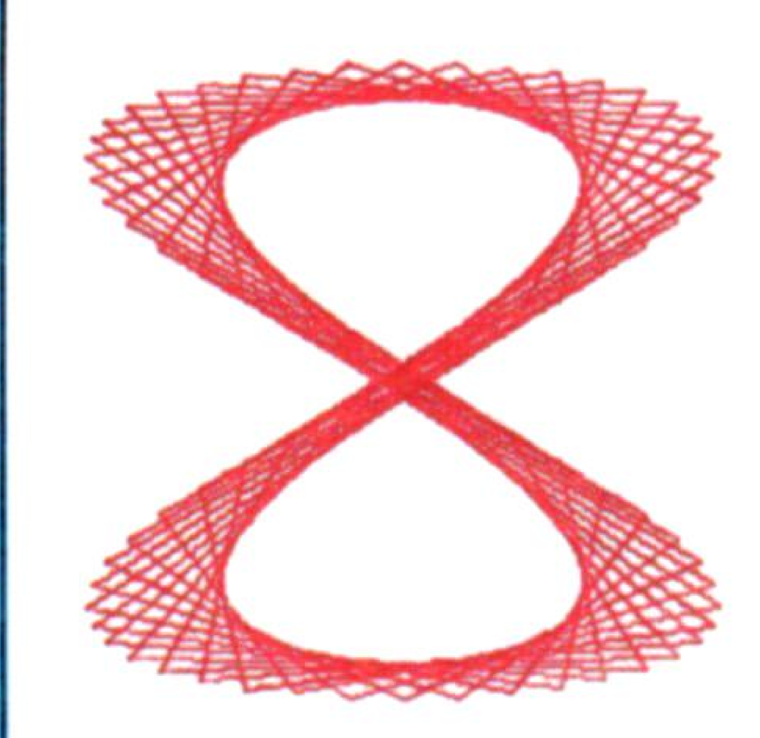
1. Juego de conversión para utilizar control por teclado:
Cambie ESTABLECER.DIABLOS MIRAR,VERIFICAR. Elimine ENCPAL. Agregue MOVER y TECLALEIDA.

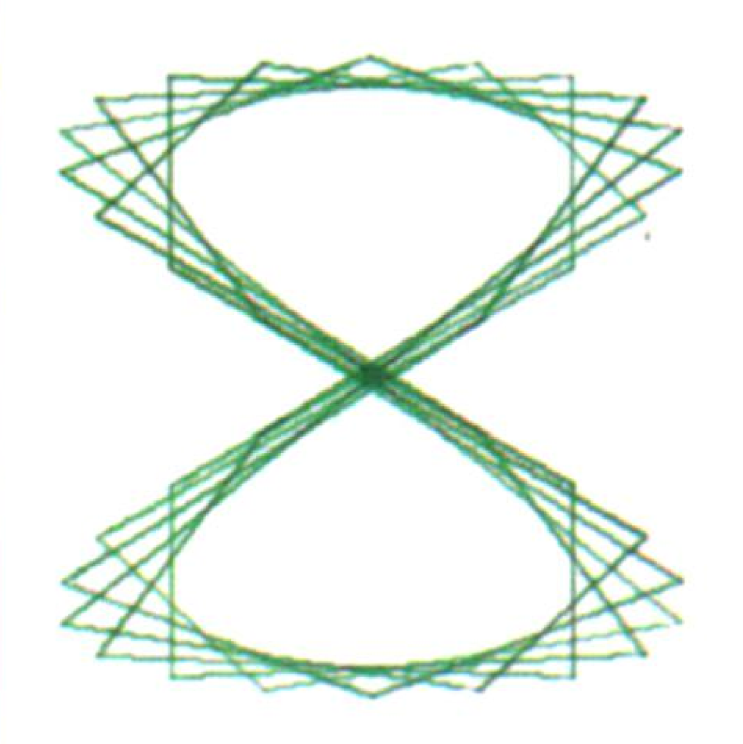
```
TO ESTABLECER.DIABLOS
   WHEN OVER :OVEJA1 :CERCA [SETSP 0]
   WHEN OVER :OVEJA2 :CERCA [SETSP 0]
   WHEN TOUCHING :OVEJA1 :OVEJA2 [SETSP 0]
   WHEN TOUCHING :PERRO :OVEJA1 [SETSP 0]
   WHEN TOUCHING :PERRO :OVEJA2 [SETSP 0]
END
TO MIRAR
   MOVER TECLALEIDA
   IF :VELOCIDAD = 0 [VERIFICAR]
   MIRAR
END
TO VERIFICAR
   IF COND OVER :OVEJA1 :CERCA [ASK
      :OVEJA1 [BK 10 RT 90]]
   IF COND OVER :OVEJA2 :CERCA [ASK
      :OVEJA2 [BK 10 RT 90]]
   IF COND TOUCHING :OVEJA1 :OVEJA2
[CHOQUE]
   IF COND TOUCHING :PERRO :OVEJA1 [ASK
      :OVEJA1 [RT 90]]
   IF COND TOUCHING :PERRO :OVEJA2 [ASK
      :OVEJA2 [RT 90]]
   ESTABLECER.VELOCIDADES
END
TO MOVER :COM
   IF :COM = "W [ASK :PERRO [SETH 0]]
   IF :COM = "S [ASK :PERRO [SETH 90]]
   IF :COM = "Z [ASK :PERRO [SETH 180]]
   IF :COM = "A [ASK :PERRO [SETH 270]]
   IF :COM = "Q [ASK :TORTUGA [DIBUJAR.
      CAJA]]
END
TO TECLALEIDA
   IF KEYP [OUTPUT RC]
   OUTPUT"
END
```

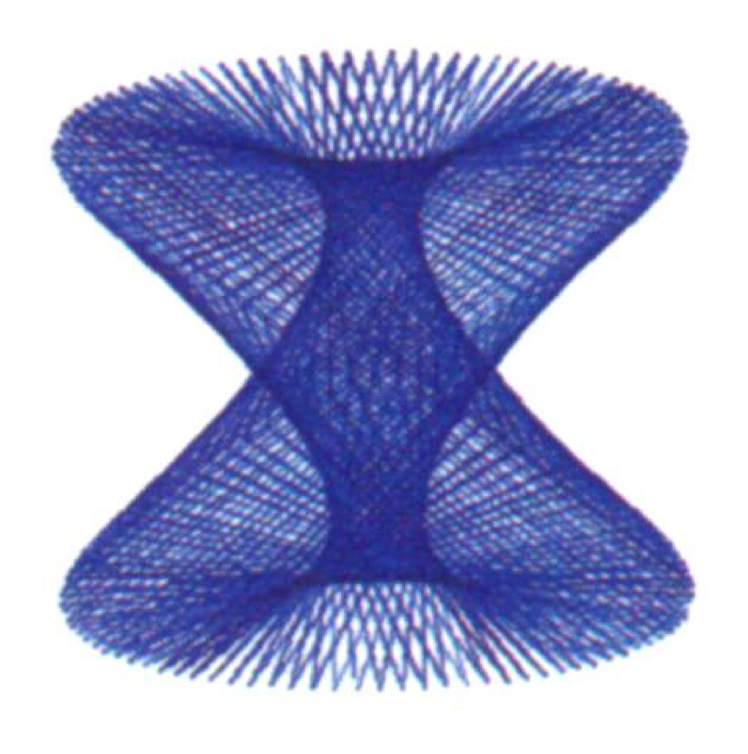
2. El juego de los meteoritos: defina forma 1 como un meteorito y forma 2 como la nave espacial.

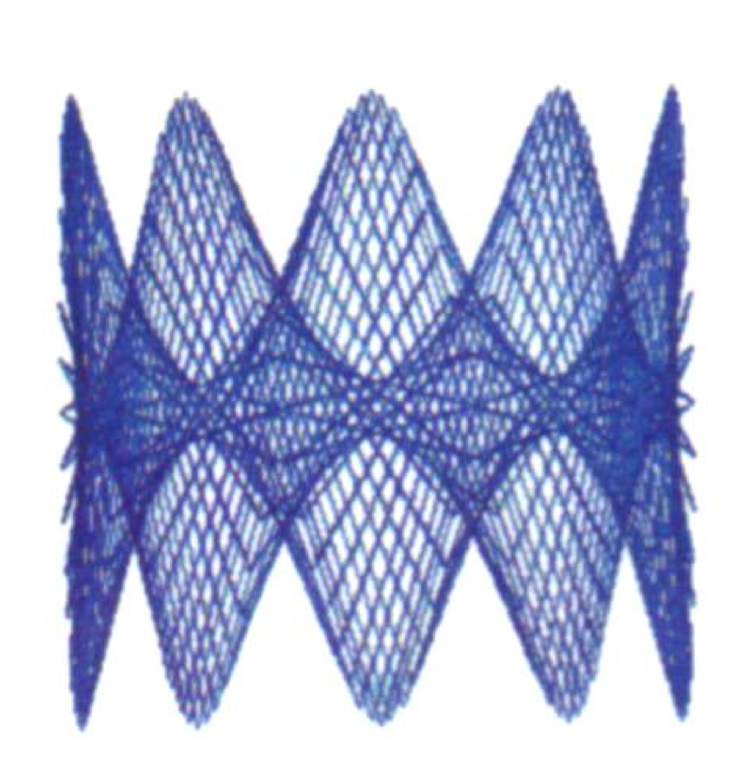
```
TO JUGAR
   CS FS
   EST 0 1 [– 100 80] 180 199
   EST 1 1 [0 80] 180 199
   EST 2 1 [100 90] 180 199
   EST 3 2 [0 – 80] 90 50
   ESTABLECER.DIABLOS
   MOVER.AZAR 0
END
TO EST :NUM :FORMA :POS :CABEZA :VEL
   TELL :NUM SETSH :FORMA
   PU SETPOS :POS
   SETH :CABEZA ST SETSP :VEL
END
TO ESTABLECER. DIABLOS
   WHEN TOUCHING 0 3 [BANG]
   WHEN TOUCHING 1 3 [BANG]
   WHEN TOUCHING 2 3 [BANG]
   WHEN 15 [ENCPAL]
END
TO BANG
   TELL [0 1 2 3]
   SETSP 0 SS
   PRINT " PRINT "
   PRINT "SALPICADO
END
TO ENCPAL
   IF (JOY 1) < 0 [STOP]
   ASK 3 [SETH 45* JOY 1]
END
TO MOVER.AZAR :NUM
   IF SPEED = 0 [(PRINT "MARCADOR :NUM)
      STOP]
   ASK RANDOM 3 [SETH 145 + RANDOM 70]
   MOVER.AZAR :NUM+1
END
```

FIGURAS LISSAJOUS









Saber diseñar el programa

Hasta ahora nos hemos limitado a construir rutinas sencillas. Es el momento de abordar tareas más ambiciosas, como la estructuración de programas en assembly

ESPECIFICACIÓN
DISEÑO
CODIFICACIÓN
DEPURACIÓN
RETURN
COMPROBACIÓN
DOCUMENTACIÓN
MANTENIMIENTO

Kevin Jones

El diseño importa
Resultan difíciles de observar las reglas de una buena estructura cuando se programa en lenguaje máquina. Pero no es difícil crear un programa según las reglas de un buen diseño, además de ser un instrumento excelente de claridad y de ahorro de tiempo en la depuración

Mucho llevamos escrito sobre las ventajas de un adecuado diseño de los programas, la construcción por módulos y la programación estructurada en el contexto de los lenguajes de alto nivel. Pero tanto las dificultades como las ventajas se encuentran agrandadas en los de bajo nivel. No existen en assembly estructuras apropiadas de control como las hay en BASIC (p. ej., WHILE...WEND, IF...THEN...ELSE) que le proporcionen algún tipo de estructura al código. Tampoco se dispone de notaciones adecuadas, diversos tipos de variables, y ni siquiera se puede pedir a un programa en assembly que no sea seis o diez veces mayor, en número de instrucciones, que el mismo programa escrito en un lenguaje de alto nivel. Y encima, es mucho más fácil que se cometan errores de consecuencias desastrosas, tales como borrar todos los datos introducidos en un disco por culpa de un mínimo error existente en un solo byte. Con el fin de que la programación en assembly resulte menos aventurada, vamos a exponer un método para abordarla.

No hay nada absolutamente nuevo en la programación estructurada o ingeniería de software: un programador experimentado sabe sencillamente que tener el problema totalmente especificado y la claridad en el análisis constituyen la base ineludible de un estilo de programación con éxito. Nada más ilustrativo para probar esto que tratar de descubrir errores en un programa escrito en lenguaje máquina por usted de una manera poco estructurada e indocumentada y que dejó en algún sitio hace algunos meses. Un buen diseño y unos adecuados métodos de trabajo garantizan por sí solos un buen programa.

Fases en el diseño del programa

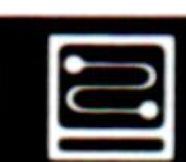
- *Especificar el problema.* En esta fase, el programador debe cuidar sobre todo de especificar cuáles son las entradas y cuáles las salidas. A menudo los dispositivos periféricos son controlados directamente (en especial, el teclado y la pantalla), por lo que se han de tener en cuenta las señales utilizadas. Puede que también existan limitaciones de tiempo. Puede que usted carezca de rutinas apropiadas para convertir una cadena de bytes que entra o sale en la forma en que el programa lee los datos (p. ej., la conversión de una cadena de caracteres ASCII en un número decimal en formato binario). Es, pues, importante especificar no sólo el formato en que surge el dato sino también el que será exigido por el resto del programa.

- *Diseño del programa.* Se trata ahora de idear los procesos por los que la entrada especificada alcanza la salida o resultado deseado. Tales procesos han de agruparse por módulos en la medida de lo posible, autosuficientes desde un punto de vista lógico, junto con los datos que cada proceso necesita. Hay dos técnicas principales para "cortar" un programa en módulos: 1) La técnica *bottom-up* (de abajo arriba), en la que se hace una recolección de módulos que pueden resultar útiles en el contexto del programa para después ajustarlos; 2) La técnica del *top-down* (de arriba abajo), en la que se va descomponiendo sucesivamente el programa en unidades cada vez más pequeñas, interesándonos antes en la función a desempeñar que en cómo lograrla, hasta el punto en que el proceso no puede seguir más adelante. Sólo en ese momento se comienza a pensar en el modo de codificar cada módulo.

El diseño *bottom-up* ofrece la ventaja de emplear módulos de librería, fáciles de conjuntar y cuyo resultado suele ser un uso más eficaz de la memoria. La desventaja está en que el programa en su totalidad puede resultar difícil de depurar y comprobar, así como puede que no sea tan comprensible. El diseño *top-down* proporciona programas mejor estructurados, y cada fase del proceso puede ser comprobada por separado por medio de "colillas" (*stubs*), breves rutinas que reemplazan los módulos todavía por escribir limitándose a aceptar entradas y proporcionando una salida correcta sin realizar proceso alguno. La desventaja se halla en que los programas probablemente ocuparán mucha más memoria y difícilmente las rutinas creadas podrán tener un uso inmediato en otro lugar.

Dentro de cada módulo se especificarán los datos que se necesitan, sus estructuras y algoritmos. En este nivel resulta útil un diagrama de flujo para representar los algoritmos, pero hay quienes prefieren, por más sencillo, trabajar libremente con un lenguaje de alto nivel llamado seudocódigo. La base de este seudocódigo suele ser el PASCAL, pero no hay razón alguna para prescindir del BASIC. Éste nos permite diseñar algoritmos y datos de una manera que nos es familiar, y relega el trabajo de bajo nivel a las tareas más sencillas de traducir a assembly los algoritmos en seudocódigo. Lo cual es mucho más fácil que intentar diseñar y codificar en assembly al mismo tiempo.

- *Codificación.* Si las rutinas fueron correctamente diseñadas, ésta será sin duda la fase más sencilla y



breve de todas. Para traducir un algoritmo en alto nivel a código en bajo nivel es esencial que se trasladen al bajo nivel las estructuras de control empleadas en alto nivel, desechando la tentación de emplear BRA y JMP indiscriminadamente. Recuerde que el tiempo que ahorra escribiendo código no estructurado se va a derrochar en una siguiente fase de depuración con sus frustrantes procesos de "ensayo y error".

En el diagrama de esta página proporcionamos algunos ejemplos de cómo pueden codificarse las más comunes estructuras de control, suponiendo, para mayor simplicidad, que los ítems de datos empleados son de ocho bits.

El problema de una codificación de este tipo de las estructuras de control es que el programa puede alargarse más de lo previsto. Esto es irrelevante si no se tiene limitación de memoria; una codificación más breve no siempre significa menor tiempo de ejecución, sino un desarrollo más lento y mucho tiempo invertido en la depuración. Pero si el espacio está limitado, es mejor escribir en una forma estructurada espaciosa e intercalar una fase de optimización en la que la codificación operativa puede abreviarse teniendo en cuenta situaciones particulares y respetando en lo posible la estructura esencial.

- *Depuración.* En esta fase se probará cada módulo por separado (empleando *stubs* donde sea necesario) para asegurarse que proporciona las salidas adecuadas a las entradas válidas. La depuración de programas en assembly dista mucho de parecerse a la del BASIC. Para poder observar lo que está sucediendo, es necesario inspeccionar el contenido de los registros y de las posiciones de memoria empleadas por el programa, y cambiarlo si llega el caso. Es casi imposible depurar un programa assembly sin la ayuda de una utilidad que permita poner y sacar puntos de ruptura (*breakpoints*). Estos puntos permiten ejecutar un programa hasta dicha ruptura, volcar los registros e inspeccionar y cambiar los contenidos de la memoria.

- *Comprobación.* Una vez comprobado cada módulo, hay que acoplar el programa entero y probarlo con datos apropiados. Esto es todavía más fácil cuando se sabe que cada una de las partes funciona correctamente.

- *Documentación.* Es importante, dado que los programas en assembly son más difíciles de comprender que los escritos en lenguajes de alto nivel. Es imprescindible, en concreto, documentar el uso de la memoria, de la pila (sobre todo al pasar parámetros) y de los registros dentro de las subrutinas.

- *Mantenimiento.* Si un programa va a ser utilizado después de un buen período de tiempo, de seguro necesitará alguna revisión, ya sea para enmendar errores, ya sea para hacer alguna que otra mejora. Aquí, en esta fase, es donde se valora el tiempo invertido en un diseño cuidadoso y en una buena documentación. Si el programa se diseñó y/o se documentó pobremente, será mejor que lo vuelva a escribir por completo y no intentar cambios.

Necesitamos ahora un proyecto para aplicar estas habilidades. Nada más apropiado, para nuestra primera aventura en assembly estructurado, que un monitor/depurador en código máquina. Si ya ha usado un ensamblador, le resultará familiar el tipo de utilidades que se esperan de un monitor/depurador. En esencia, proporcionará al programador el tipo de facilidades de edición que el programador en BASIC da por supuestas; es decir, la posibilidad de inspeccionar y cambiar los contenidos de la memoria.

En el próximo capítulo de este curso de lenguaje máquina trataremos este proyecto y recorreremos, desde la fase de diseño, todas las etapas descritas anteriormente, con la finalidad de crear una ayuda de programación importante y que le sea de auténtica utilidad.

Columna vertebral
No existen estructuras de control escritas en assembly, por eso puede ser útil imitar los métodos comprobados en lenguajes de alto nivel. Las estructuras que aquí mostramos son claras y elegantes en ambos lenguajes, de bajo y alto nivel, y se emplearán para excluir cualquier otra alternativa

Estructuras de control

Estructura IF...THEN...ELSE

Seudocódigo	Lenguaje assembly
	TRES FCB 3
	....
IF NUM1 = 3	IF LDA NUM1
THEN	CMPA TRES
rutina1	BNE ELSE
	THEN ....
ELSE	*rutina1
rutina2	BRA FINIF
ENDIF	ELSE ....
	*rutina2
	FINIF ....

Estructura WHILE...WEND

Seudocódigo	Lenguaje assembly
WHILE NUM1 <=3	WHILE LDA NUM1
rutina a repetir	CMPA TRES
WEND	BGT FINWHILE
	*rutina a repetir
	BRA WHILE
	FINWHILE ....

Estructura REPEAT...UNTIL

Seudocódigo	Lenguaje assembly
REPEAT	REPEAT...............
rutina a repetir	*rutina a repetir
UNTIL NUM1 <3	LDA NUM1
	CMPA TRES
	BGE REPEAT
	UNTIL ....

Estructura FOR...NEXT

Seudocódigo	Lenguaje assembly
FOR NUM1=1 TO NUM2	LDA NUM2
rutina a repetir	FOR ....
NEXT NUM1	*rutina a repetir
	DECA
	BGT FOR
	NEXT ....

El efecto del defecto

"Deus ex machina" es un original juego que le permite al usuario asumir el papel de protagonista de una "fantasía televisiva totalmente animada"

Dado que los juegos por ordenador se han convertido en una parte importantísima de la industria del ocio, quizá era inevitable que las firmas de software aunaran sus esfuerzos con otros sectores del negocio del entretenimiento. Automata Software, empresa conocida por su serie de juegos de Piman, ha dado los primeros pasos en este camino desarrollando un producto que no sólo contiene software para ordenadores sino también una cassette de audio que se puede sincronizar con el programa para ordenador para proporcionarle al juego una banda sonora.

La idea sobre la que se sustenta *Deus ex machina*, cuyo desarrollo consumió seis meses y su programación otros tres, es que un ordenador omnipotente del futuro se rebela y ayuda a crear un "defecto" humano, un ser imperfecto. El jugador, como ser imperfecto, pasa a través de diversas etapas a lo largo del juego, que describen experiencias que abarcan desde la infancia hasta la vejez. La participación directa del jugador comienza en la concepción, guiando al espermatozoide hacia el óvulo. A medida que el niño va creciendo, está sujeto al ataque constante de la "policía del defecto". Está en manos del jugador desviar estos ataques, utilizando ya sea el teclado o la palanca de mando. Se consiguen puntos manteniendo el "porcentaje de entidad ideal", que empieza con un 99 % y va disminuyendo en función de los asaltos de la policía del defecto. Cuando el ser imperfecto alcanza la edad adulta, la naturaleza de esos ataques cambia y el jugador se debe adaptar a ellos.

Después de cargado el juego, se debe sincronizar la banda sonora con el programa. Al hacerlo se debe tener mucho cuidado, porque las distintas pantallas están cronometradas como para coincidir exactamente con las palabras y la música, y esto contribuye enormemente al entretenimiento que proporciona el paquete.

Desarrollo del juego
Deus ex machina es para jugar con él o para contemplarlo como fuente de entretenimiento. Posee una amplia variedad de visualizaciones, si bien algunas guardan cierto parecido entre sí. Las tácticas requeridas para mantener la "entidad ideal" cambian constantemente. El marcador se indica como porcentaje en el ángulo inferior derecho y va disminuyendo lentamente a medida que el juego (y la vida del ser imperfecto) avanza

El tanque de interrogación

La incubadora

Ian McKinnell

El programa se divide en dos segmentos, una mitad en cada lado de la cassette, que representan un total de 96 K de código. Al final del lado 1, después de una escena amorosa en la que el jugador debe desplazar el cursor por su cuerpo para recibir los besos que van flotando hacia él, se debe cargar el lado 2. No se debe apagar el ordenador y, nuevamente, debe ponerse atención en sincronizar la banda de sonido correctamente. La implicación del jugador en la segunda mitad consiste fundamentalmente en ir saltando obstáculos antes de llegar a la "vejez". En este punto aparecen en la pantalla grandes coágulos de sangre, que el jugador debe disolver. Al concluir el juego, independientemente del marcador, el ser imperfecto muere.

Deus ex machina es inusual por el hecho de que no hay ningún marcador ganador y, en realidad, el jugador ni siquiera necesita participar para nada en el juego. Los acontecimientos se suceden de la misma forma sin ninguna participación, de modo que el jugador tiene la opción de involucrarse en el desarrollo del juego o bien mirarlo como una fuente de entretenimiento. Los gráficos son excelentes e imaginativos. A pesar de que no hay ninguna pantalla que corte el aliento, sí reflejan todas el cuidado y la atención que se le dedicaron a todos los detalles del paquete en su conjunto.

La música de la banda sonora la escribió y la dirigió totalmente el cofundador de Automata, Mel Croucher, quien también escribió el guión. Las canciones son agradables, sin llegar a ser excepcionales. El mejor número es el que acompaña a la escena en el cual el ser imperfecto cobra vida, y lo interpreta Ian Dury.

El guión y la banda sonora son muy diferentes a los de la mayoría de los juegos por ordenador y reflejan la filosofía de no violencia a la que responden todos los juegos de Automata. Los usuarios que disfruten destruyendo hordas frenéticas de extraterrestres atacantes probablemente quedarán decepcionados, y a muchas personas el contenido semimístico de las letras de las canciones les resultará molesto. Sin embargo, se debe aplaudir de todo corazón a Automata por su innovadora idea. El programa es un valiente experimento y sin ninguna duda será considerado como un paso importante en el desarrollo del ocio informatizado.

Deus ex machina: Para el Spectrum de 48 K
Editado por: Automata Ltd, 27 Highland Road, Portsmouth, Hants, PO4 9DA, Gran Bretaña
Autores: Mel Croucher, Andrew Stagg
Palanca de mando: Opcional
Formato: Cassette



Editorial Delta, S.A.

